Vivado Design Suite User Guide

Embedded Processor Hardware Design

UG898 (v2013.4) December 18, 2013

XILINX®

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/2013	2013.1	New release for Vivado® Design Suite version 2013.1.
06/19/2013	2013.2	New section: Zynq-7000 Processing System Simulation, page 31 New chapter: Designing with the MIG Core New chapter: Chapter 5, Reset and Clock Topologies in IP Integrator
10/2/2013	2013.3	Minor edits to the document.
12/18/2013	2013.4	Minor edits to the document.

Table of Contents

Appendix A: Additional Resources

Chapter 1

Introduction

Overview

This chapter provides an introduction to using the Xilinx® Vivado® Design Suite flow for programming an embedded design using the Zynq®-7000 All Programmable (AP) SoC device or the MicroBlaze™ processor.

Embedded systems are complex. Hardware and software portions of an embedded design are projects in themselves. Merging the two design components so that they function as one system creates additional challenges. Add an FPGA design project, and the situation can become very complicated.

To simplify the design process, Xilinx offers several sets of tools. It is a good idea to know the basic tool names, project file names, and acronyms for these tools, which can be found in the Xilinx Glossary.

The Vivado Integrated Design Environment (IDE) includes the IP integrator tool, which you can use to *stitch* together a processor-based design. This tool, combined with the Xilinx Software Development Kit (SDK), provides an integrated environment to design and debug microprocessor-based systems and embedded software applications.

Hardware and Software Tool Flow Overview

The Vivado tools provide specific flows for programming, based on the processor. The Vivado IDE uses the IP integrator with graphic connectivity screens to specify the device, select peripherals, and configure hardware settings.

The Zynq-7000 AP SoC uses the Vivado IP integrator to capture hardware platform information in XML format applications, along with other data files. These are used in software design tools to create and configure Board Support Package (BSP) libraries, infer compiler options, program the PL, define JTAG settings, and automate other operations that require information about the hardware. The Zynq-7000 SoC solution reduces the complexity of an embedded design by offering an ARM Cortex A9 dual core as an embedded block, and programmable logic along with it, on a single SoC.

Xilinx provides the following design tools for developing and debugging software applications for Zynq-7000 AP SoC and MicroBlaze processor devices:

- Software IDE
- GNU-based compiler toolchain
- JTAG debugger

These tools let you develop both bare-metal applications that do not require an operating system, and applications for the open-source Linux operating system. The Vivado IP integrator captures information about the Processing System (PS) and peripherals, including configuration settings, register memory map, and the bitstream for Processing Logic (PL) initialization.

Software solutions are also available from third-party sources that support Cortex-A9 processors, including but not limited to:

- Software IDEs
- Compiler toolchains
- Debug and trace tools
- Embedded OS and software libraries
- Simulators
- Models and virtual prototyping tools

Third-party tool solutions vary in the level of integration and direct support for Zynq-7000 devices.

See the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 1] for more information about the SDK and programming for Zynq devices. The SDK is a standalone product, and is available for download from www.xilinx.com.

Figure 1-1 illustrates the tools flow for embedded hardware.

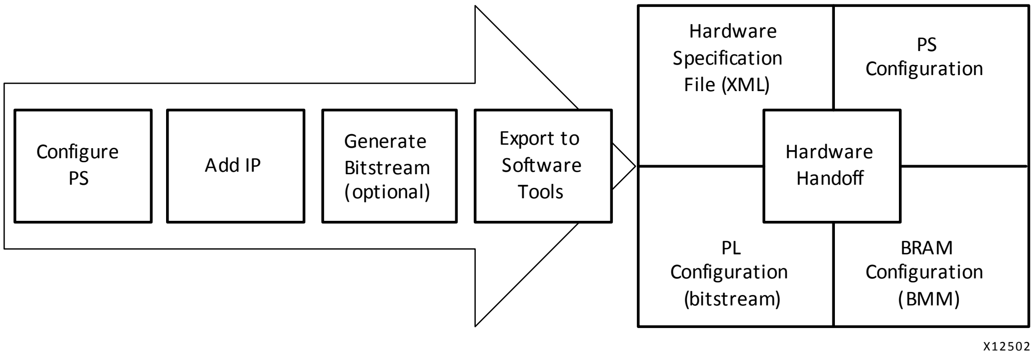


Figure 1-1: **Hardware Design Tool Handoff to Software Tools**

To start a Zynq-7000-based design, do the following:

1. Create a new Vivado IDE project.
2. Create a block design in the IP Integrator tool and start instantiating the Zynq Processing System 7 IP along with any other Xilinx IP or your custom IP.
3. Run the design through synthesis and implementation and export the hardware to SDK.
4. In SDK, you create your software application, which you can then program into the target board.

Chapter 2

Using a Zynq-7000 Processor in an Embedded Design

Introduction

This chapter describes how to use the Xilinx® Vivado® Design Suite flow for using the Zynq®-7000 All Programmable (AP) SoC device.

The examples target the Xilinx ZC702 Rev 1.0 evaluation board and the tool versions in the 2013.2 Vivado Design Suite release.

IMPORTANT: *The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for embedded processor designs, including designs targeting Zynq devices and MicroBlaze™ processors. XPS only supports designs targeting MicroBlaze processors. Both IP integrator and XPS are available from the Vivado IDE.*

Designing for Zynq-7000 Devices in the Vivado IDE

Designing for Zynq-7000 AP SoC devices is different using the Vivado IDE than in the ISE® Design Suite and Embedded Development Kit (EDK).

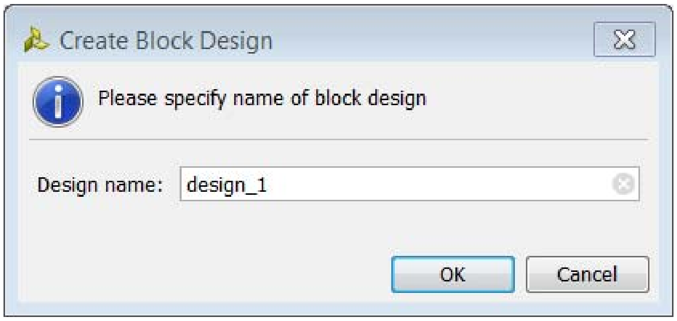
The Vivado IDE uses the IP integrator tool for embedded development. The IP integrator is a GUI-based interface that lets you stitch together complex IP subsystems.

A variety of IP are available in the Vivado IDE IP Catalog to accommodate complex designs.

You can also add custom IP to the IP Catalog. See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 2] for more information.

Creating an IP Integrator Design with the Zynq-7000 Processor

Click the IP integrator **Create Block Design** button Create Block Design to open the Create Block Design dialog box, where you can enter the Design Name as shown in the figure below.



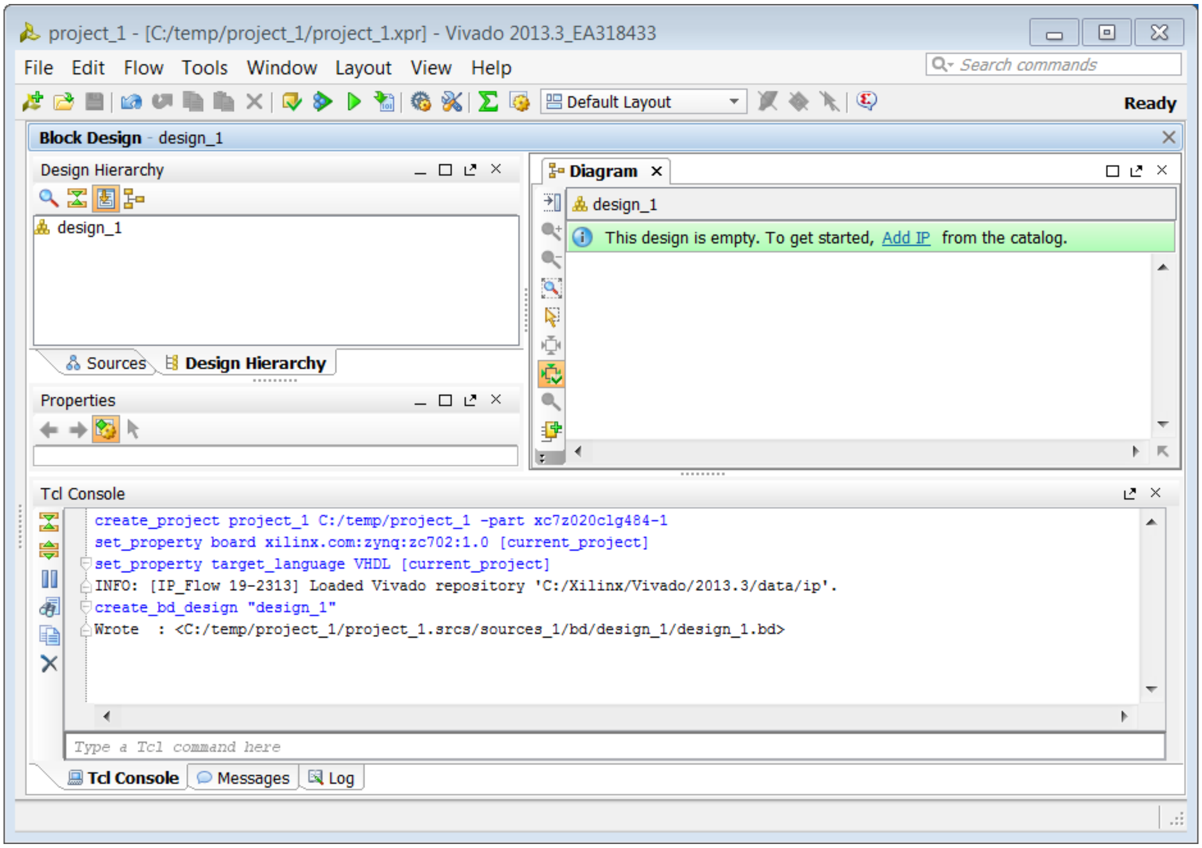
Create Block Design

Please specify name of block design

Design name: design_1

Figure 2-1: **Design Name Dialog Box**

The Block Design window opens, as shown in Figure 2-1.



project_1 - [C:/temp/project_1/project_1.xpr] - Vivado 2013.3_EA318433

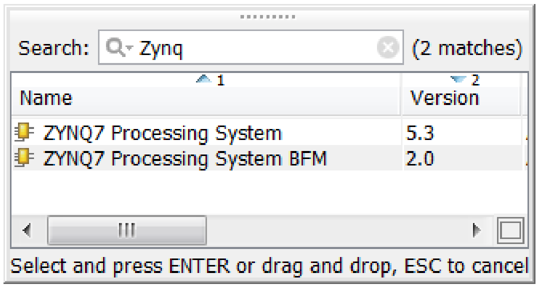
This design is empty. To get started, Add IP from the catalog.

Tcl Console

```
create_project project_1 C:/temp/project_1 -part xc7z020clg484-1
set_property board xilinx.com:zynq:zc702:1.0 [current_project]
set_property target_language VHDL [current_project]
INFO: [IP_Flow 19-2313] Loaded Vivado repository 'C:/Xilinx/Vivado/2013.3/data/ip'.
create_bd_design "design_1"
Wrote  : <C:/temp/project_1/project_1.srcs/sources_1/bd/design_1/design_1.bd>
```

Figure 2-2: **Block Design Window**

1. In the empty design, there is an option to **Add IP** from the IP Catalog. You can also right-click in the canvas to add IP from the popup menu.
2. Select the **Add IP** option, and a Search box opens where you can search for, and select the **ZYNQ7 Processing System**, shown in Figure 2-3.



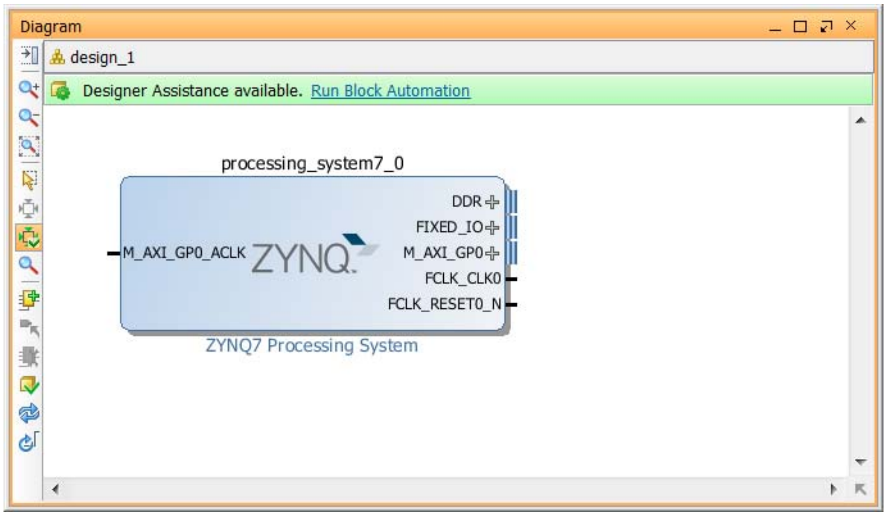
Search: Zynq (2 matches)

Name	Version
ZYNQ7 Processing System	5.3
ZYNQ7 Processing System BFM	2.0

Select and press ENTER or drag and drop, ESC to cancel

Figure 2-3: **Search IP with ZYNQ7 Processing System**

When you select the Zynq IP, the Vivado IP integrator adds the IP to the design, and a graphical representation of the processing system is displayed, as shown in Figure 2-4.



Designer Assistance available. Run Block Automation

Figure 2-4: **Graphical Display of Default ZYNQ7 Processing System**

Tcl Command:

```
create_bd_cell -type ip -vlnv
xilinx.com:ip:processing_system7:5.3 processing_system7_0
```

3. Double-click the processing system graphic to invoke the **Re-customize IP** process, which displays the Re-customize IP for the ZYNQ7 Processing System dialog box as shown in Figure 2-5.

4. Review the contents of the block design. The green colored blocks in the ZYNQ7 Processing System are configurable items. You can click a green block to open the coordinating configuration options.

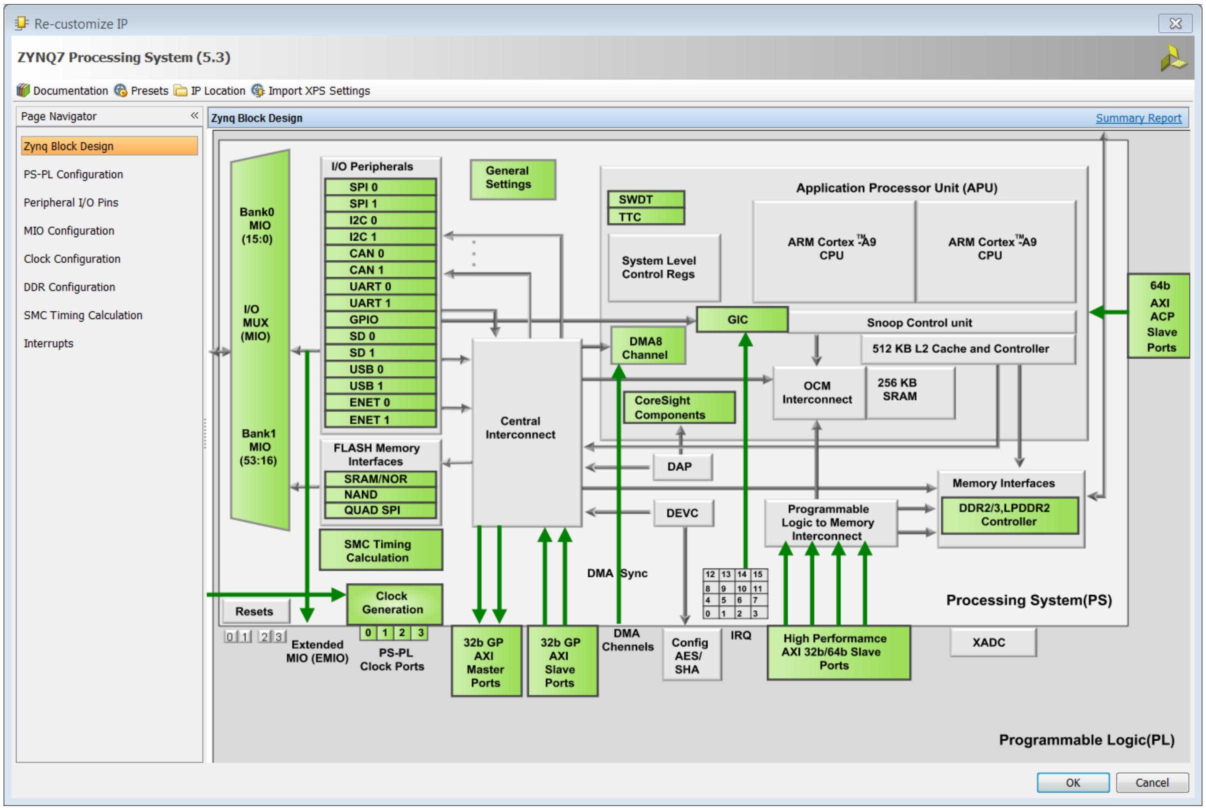


Figure 2-5: **Zynq Block Design/Configuration Dialog Box**

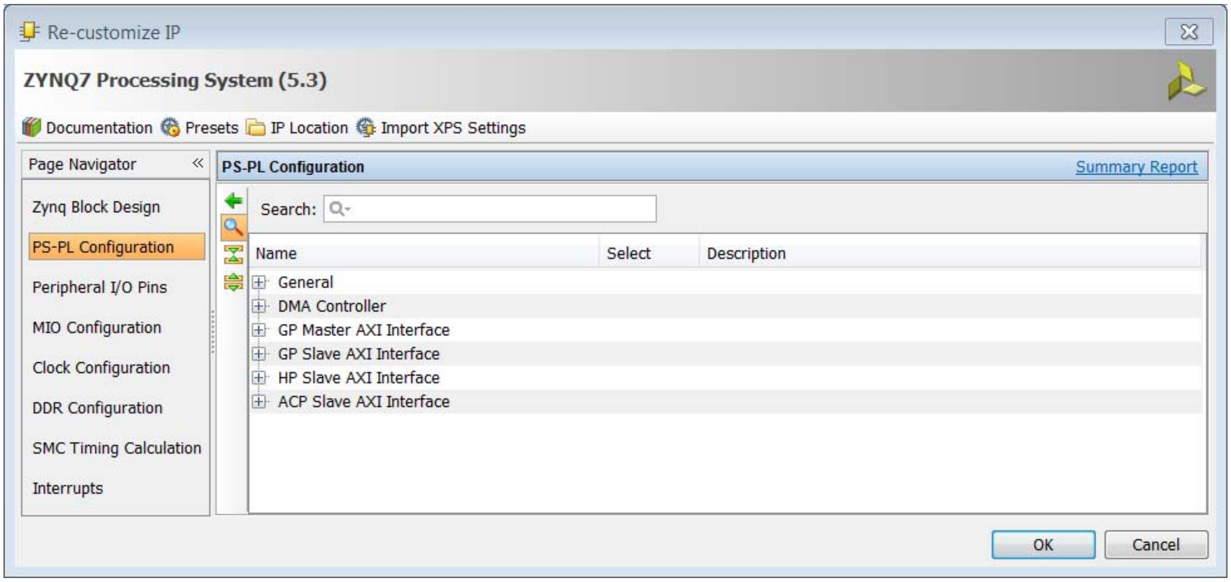
Alternatively, you can select the options from the Page Navigator on the left, as shown in Figure 2-5.

Overview of the Zynq Block Design and Configuration Window

The *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 3] provides details on the default options available in the Page Navigator. The following subsections describe in brief the Page Navigator selection options.

Processing System (PS)-Processing Logic (PL) Configuration Options

The PS-PL Configuration option tree displayed with the collapsed options as shown here.



Name	Select	Description
⊞ General		
⊞ DMA Controller		
⊞ GP Master AXI Interface		
⊞ GP Slave AXI Interface		
⊞ HP Slave AXI Interface		
⊞ ACP Slave AXI Interface		

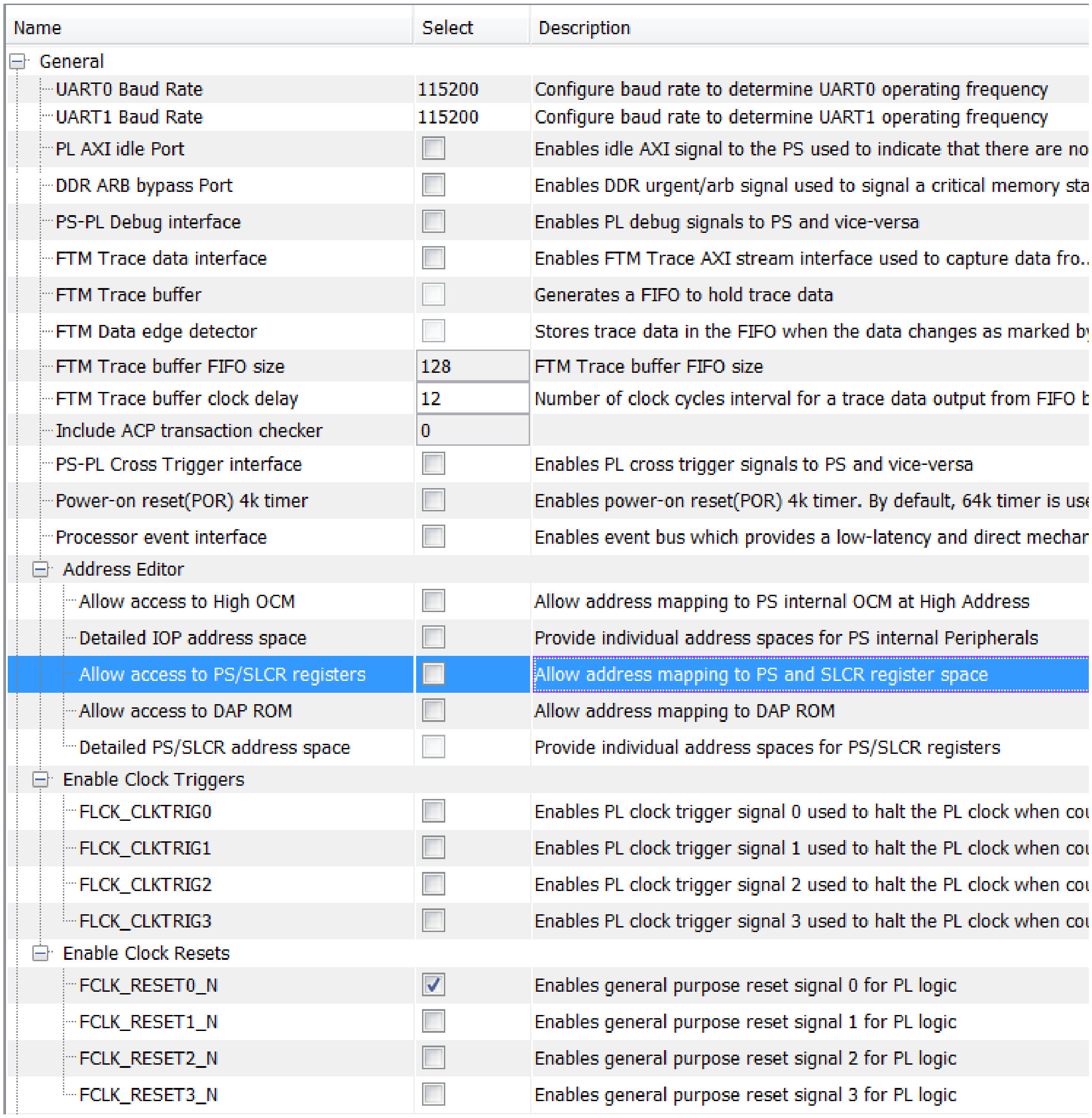
Figure 2-6: **PL-PS Configuration Pane**

Note the four buttons at the top of the window:

- **Documentation**: Click this button to open the documentation page of the Xilinx website, where you can find documentation pertaining to Zynq.
- **Presets**: Click this button to view information about the available preset options. Presets are board manager/flow specific features that let you customize the Zynq for a particular target board. The available options are MicroZed, ZC702, ZC706, and Zedboard.
- **IP Location**: This option is not available for IP integrator designs.
- **Import XPS Settings**: If you have an XML file describing the configuration of a Zynq processor from a XPS-based project, you can use this button to import that settings file to quickly configure the Zynq processor.

General Options

When you expand **General Options**, the following selections are available.

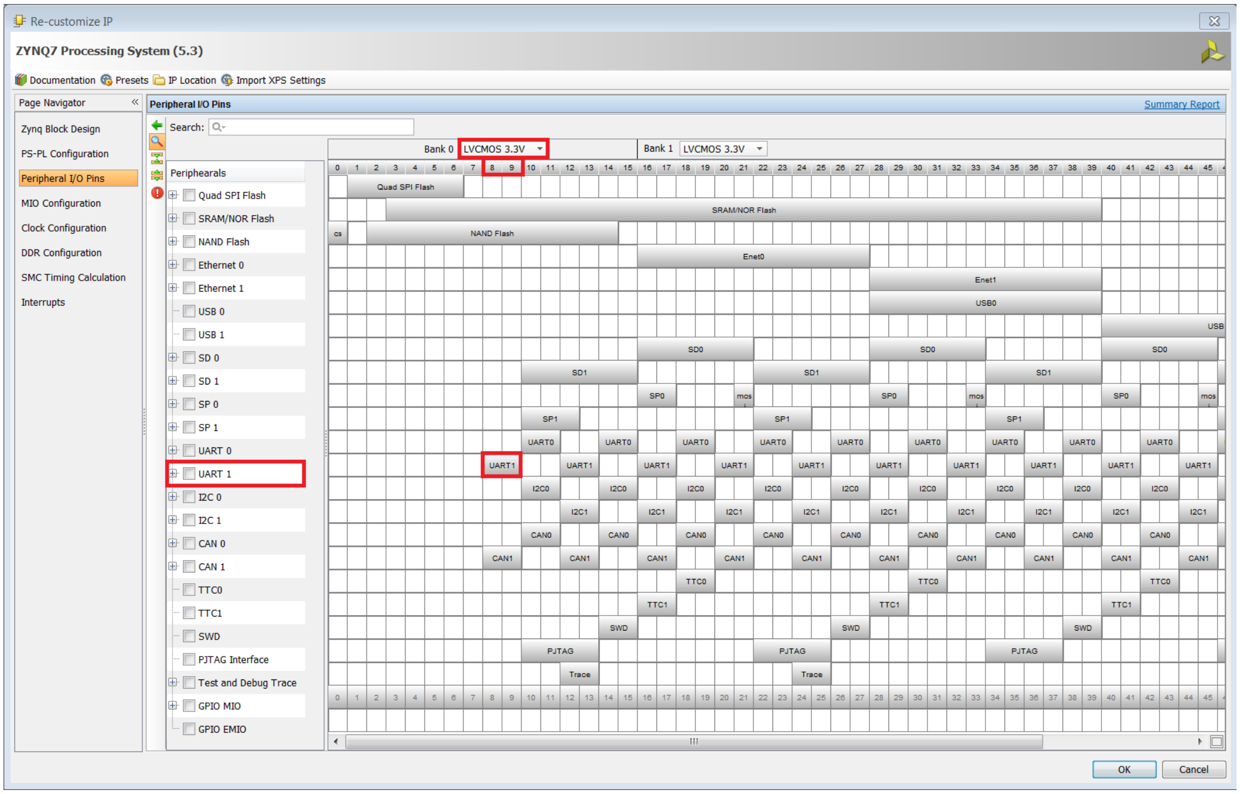


Name	Select	Description
General		
UART0 Baud Rate	115200	Configure baud rate to determine UART0 operating frequency
UART1 Baud Rate	115200	Configure baud rate to determine UART1 operating frequency
PL AXI idle Port	□	Enables idle AXI signal to the PS used to indicate that there are no
DDR ARB bypass Port	□	Enables DDR urgent/arb signal used to signal a critical memory sta
PS-PL Debug interface	□	Enables PL debug signals to PS and vice-versa
FTM Trace data interface	□	Enables FTM Trace AXI stream interface used to capture data fro..
FTM Trace buffer	□	Generates a FIFO to hold trace data
FTM Data edge detector	□	Stores trace data in the FIFO when the data changes as marked by
FTM Trace buffer FIFO size	128	FTM Trace buffer FIFO size
FTM Trace buffer clock delay	12	Number of clock cycles interval for a trace data output from FIFO b
Include ACP transaction checker	0	
PS-PL Cross Trigger interface	□	Enables PL cross trigger signals to PS and vice-versa
Power-on reset(POR) 4k timer	□	Enables power-on reset(POR) 4k timer. By default, 64k timer is use
Processor event interface	□	Enables event bus which provides a low-latency and direct mechar
Address Editor		
Allow access to High OCM	□	Allow address mapping to PS internal OCM at High Address
Detailed IOP address space	□	Provide individual address spaces for PS internal Peripherals
Allow access to PS/SLCR registers	□	Allow address mapping to PS and SLCR register space
Allow access to DAP ROM	□	Allow address mapping to DAP ROM
Detailed PS/SLCR address space	□	Provide individual address spaces for PS/SLCR registers
Enable Clock Triggers		
FLCK_CLKTRIG0	□	Enables PL clock trigger signal 0 used to halt the PL clock when cou
FLCK_CLKTRIG1	□	Enables PL clock trigger signal 1 used to halt the PL clock when cou
FLCK_CLKTRIG2	□	Enables PL clock trigger signal 2 used to halt the PL clock when cou
FLCK_CLKTRIG3	□	Enables PL clock trigger signal 3 used to halt the PL clock when cou
Enable Clock Resets		
FCLK_RESET0_N	☑	Enables general purpose reset signal 0 for PL logic
FCLK_RESET1_N	□	Enables general purpose reset signal 1 for PL logic
FCLK_RESET2_N	□	Enables general purpose reset signal 2 for PL logic
FCLK_RESET3_N	□	Enables general purpose reset signal 3 for PL logic

Figure 2-7: **General Options (First Tier)**

MIO and EMIO Configuration

From the Page Navigator, you can view and configure I/O pins by either clicking on the Peripheral I/O Pins option or MIO Configuration option.



Bank 0 LVCMOS 3.3V

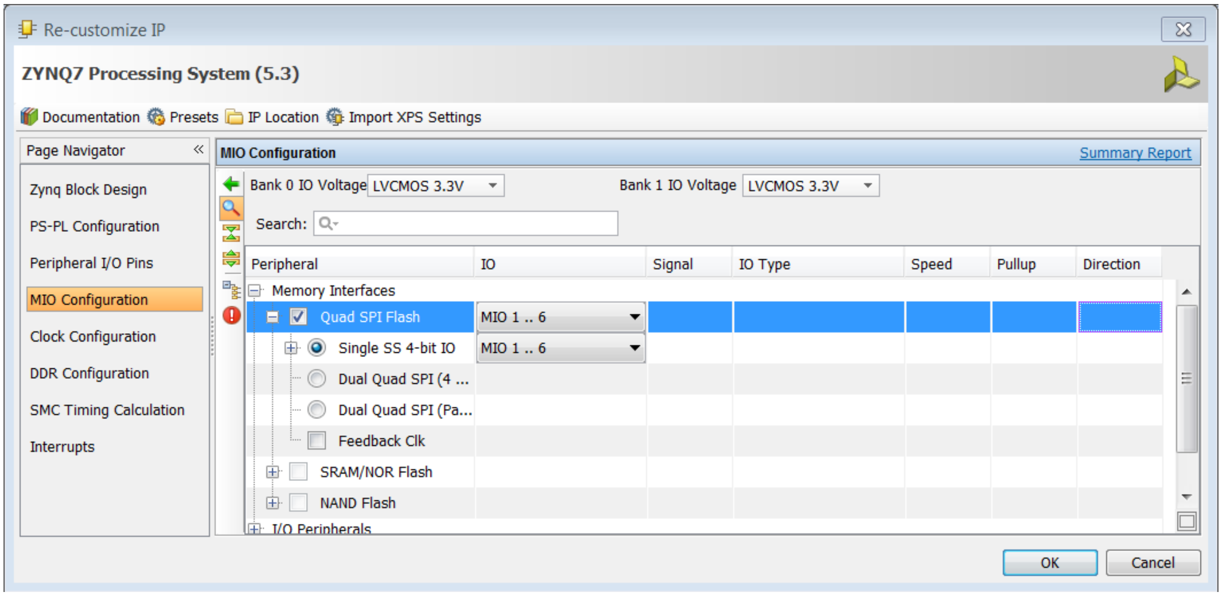
Bank 1 LVCMOS 3.3V

Figure 2-8: **Configuring Peripheral I/O Pins Using the Peripheral I/O Pins Menu**

The Zynq-7000 PS has over 20 peripherals available. You can route these peripherals directly to the dedicated Multiplexed I/Os (MIO) on the device, or through the Extended Multiplexed I/Os (EMIOs) routing to the fabric.

The configuration interface also lets you select I/O standards and slew settings for the MIO. The I/O peripheral block appears with a checkmark when you enable a peripheral. The block design depicts the status of enabled and disabled peripherals.

From the MIO Configuration option, you can do the same as shown in Figure 2-9.



Bank 0 IO Voltage LVCMOS 3.3V

Bank 1 IO Voltage LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pullup	Direction
Memory Interfaces						
☑ Quad SPI Flash	MIO 1 .. 6					
● Single SS 4-bit IO	MIO 1 .. 6					
○ Dual Quad SPI (4 ...						
○ Dual Quad SPI (Pa...						
□ Feedback Clk						
□ SRAM/NOR Flash						
□ NAND Flash						
I/O Peripherals						

Figure 2-9: **Configuring Peripheral I/O Pins Using the MIO Configuration Menu**

Chapter 2, "Signals, Interfaces, and Pins" of the *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 3] describes the MIOs and EMIOs for the 7z010 CLG225 device.

Pin Limitations

The 32 MIO pins available in the 7z010 CLG225 device restrict the functionality of the PS as follows:

- Either one USB or one Ethernet controller is available using MIO.
- Cannot boot from SDIO.
- No NOR/SRAM interfacing.
- The width of NAND Flash is limited to 8 bits.

Bank Settings

After you select peripherals, the individual I/O signals for the peripheral appear in the respective MIO locations. Use this section primarily for selecting I/O standards for the various peripherals. The PS MIO I/O buffers split into two voltage domains. Within each domain, each MIO is independently programmable.

There are two I/O voltage banks:

- Bank 0 consists of pins 0:15.
- Bank 1 consists of pins 16:53.

Each MIO pin is individually programmed for voltage signaling:

- 1.8 and 2.5/3.3 volts
- CMOS single-ended or HSTL differential receiver mode

IMPORTANT: *The entire bank must have the same voltage, but the pins can have different I/O standards.*

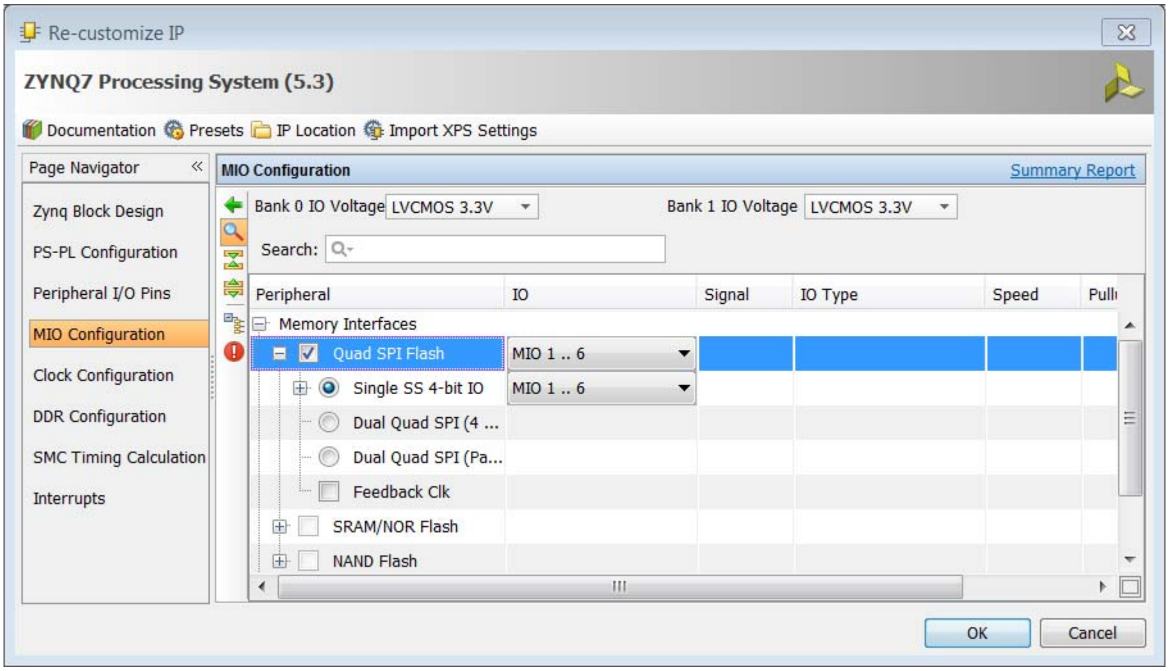
When you configure MIOs in the MIO Configuration dialog box on the Zynq tab, you can view a read-only image of the peripheral and respective MIO selections. The left side of the window lists the available peripherals. A checkmark on the peripheral indicates that a peripheral is selected.

Flash Memory Interfaces

Select one of the following in the configuration wizard:

- Quad-SPI Controller
- SRAM/NOR Controller
- AXI_HP Interfaces

Quad-SPI Controller



Bank 0 IO Voltage LVCMOS 3.3V

Bank 1 IO Voltage LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pull
⊟ Memory Interfaces					
⊟ ☑ Quad SPI Flash	MIO 1 .. 6				
⊞ ● Single SS 4-bit IO	MIO 1 .. 6				
○ Dual Quad SPI (4 ...					
○ Dual Quad SPI (Pa...					
□ Feedback Clk					
⊞ □ SRAM/NOR Flash					
⊞ □ NAND Flash					

Figure 2-10: **Quad SPI Controller Options**

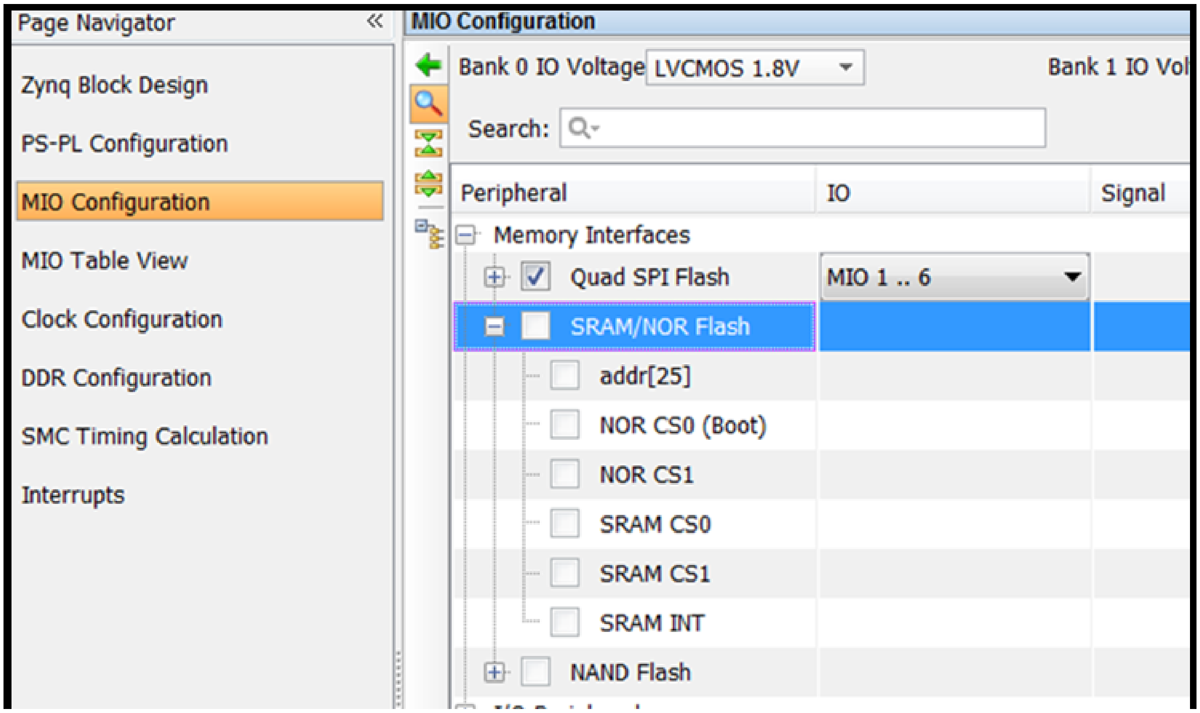
Key features of the linear Quad-SPI controller are:

- Single or dual 1x and 2x read support
- 32-bit APB 3.0 interface for I/O mode that allows full device operations including program, read, and configuration
- 32-bit AXI linear address mapping interface for read operations
- Single chip select line support
- Write protection signal support
- Four-bit bidirectional I/O signals
- Read speeds of x1, x2, and x4
- Write speeds of x1 and x4
- 100 MHz maximum Quad-SPI clock at master mode
- 252-byte entry FIFO depth to improve Quad-SPI read efficiency
- Support for Quad-SPI device up to 128 Mb density
- Support for dual Quad-SPI with two Quad-SPI devices in parallel

Additionally, the linear address mapping mode features include:

- Regular read-only memory access through the AXI interface
- Up to two SPI flash memories
- Up to 16 MB addressing space for one memory and 32 MB for two memories
- AXI read acceptance capability of four
- Both AXI incrementing and wrapping-address burst read
- Automatically converts normal memory read operation to SPI protocol, and vice versa
- Serial, Dual, and Quad-SPI modes

SRAM/NOR Controller



Bank 0 IO Voltage LVCMOS 1.8V Bank 1 IO Vol

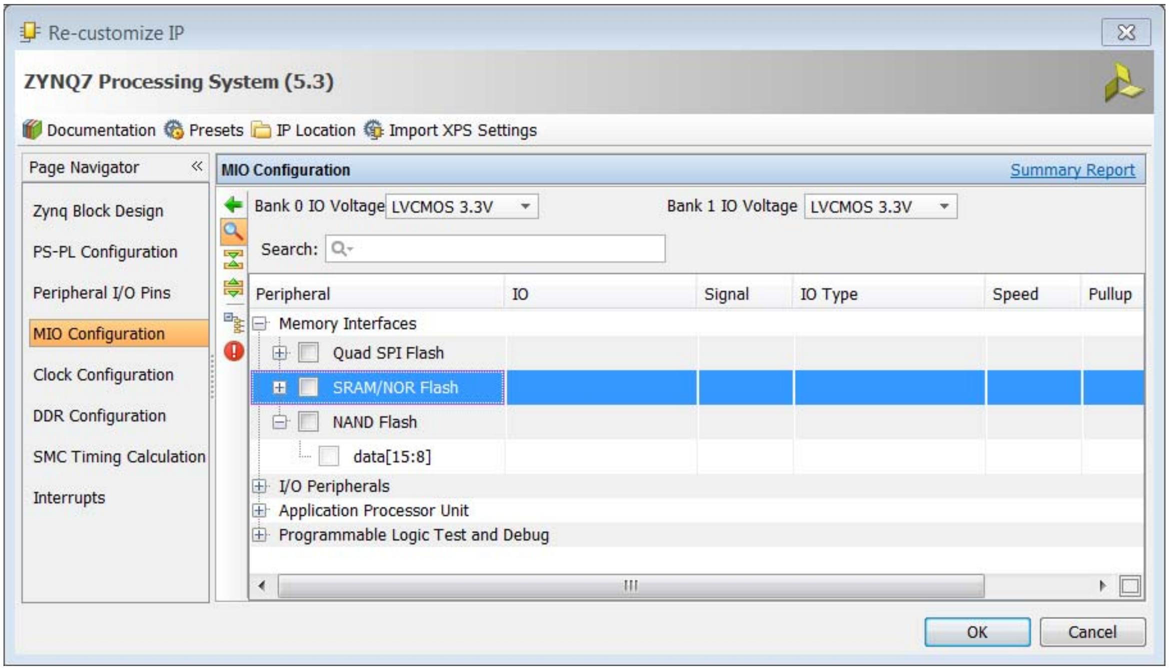
Peripheral	IO	Signal
⊟ Memory Interfaces		
⊞ ☑ Quad SPI Flash	MIO 1 .. 6	
⊟ □ SRAM/NOR Flash		
□ addr[25]		
□ NOR CS0 (Boot)		
□ NOR CS1		
□ SRAM CS0		
□ SRAM CS1		
□ SRAM INT		
⊞ □ NAND Flash		

Figure 2-11: **SRAM/NOR Flash Configuration Options**

The SRAM/NOR controller has the following features:

- 8-bit data bus width
- One chip select with up to 26 address signals (64 MB)
- Two chip selects with up to 25 address signals (32 MB + 32 MB)
- 16-word read and 16-word write data FIFOs
- 8-word command FIFO
- Programmable I/O cycle timing on a per-chip select basis
- Asynchronous memory operating mode

NAND Controller



Bank 0 IO Voltage LVCMOS 3.3V

Bank 1 IO Voltage LVCMOS 3.3V

Peripheral	IO	Signal	IO Type	Speed	Pullup
⊟ Memory Interfaces					
⊞ □ Quad SPI Flash					
⊞ □ SRAM/NOR Flash					
⊟ □ NAND Flash					
□ data[15:8]					
⊞ I/O Peripherals					
⊞ Application Processor Unit					
⊞ Programmable Logic Test and Debug					

Figure 2-12: **NAND Controller Options**

The NAND controller has the following features:

- 8/16-bit I/O width with one chip select signal
- ONFI specification 1.0
- 16-word read and 16-word write data FIFOs
- 8-word command FIFO
- Programmable I/O cycle timing
- ECC assist
- Asynchronous memory operating mode

Clock Configuration

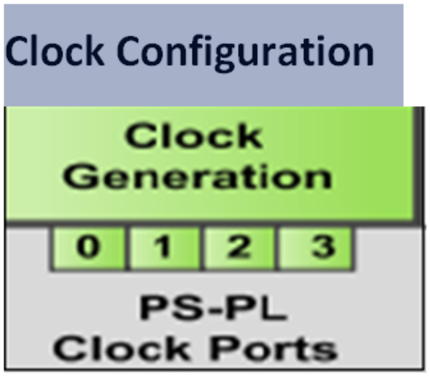
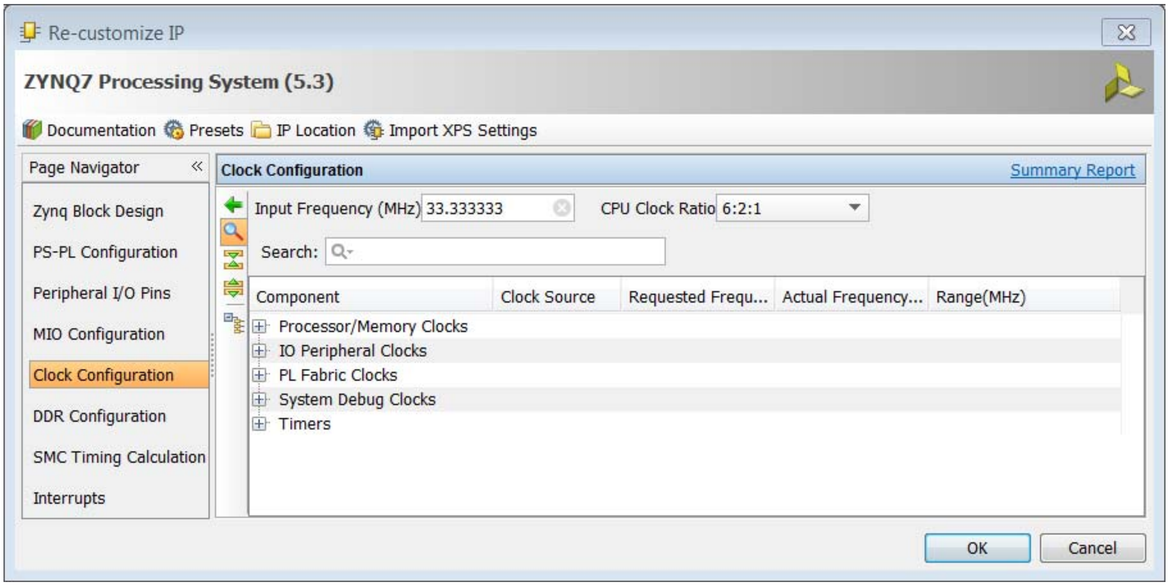


Figure 2-13: **Clock Configuration**

You can configure clocks in the Zynq-7000 device using one of the following methods:

- From the Page Navigator, click **Clock Configuration**.
- In the Zynq block design, click the **Clock Configuration** block.

Figure 2-14 shows the Clock Configuration page.



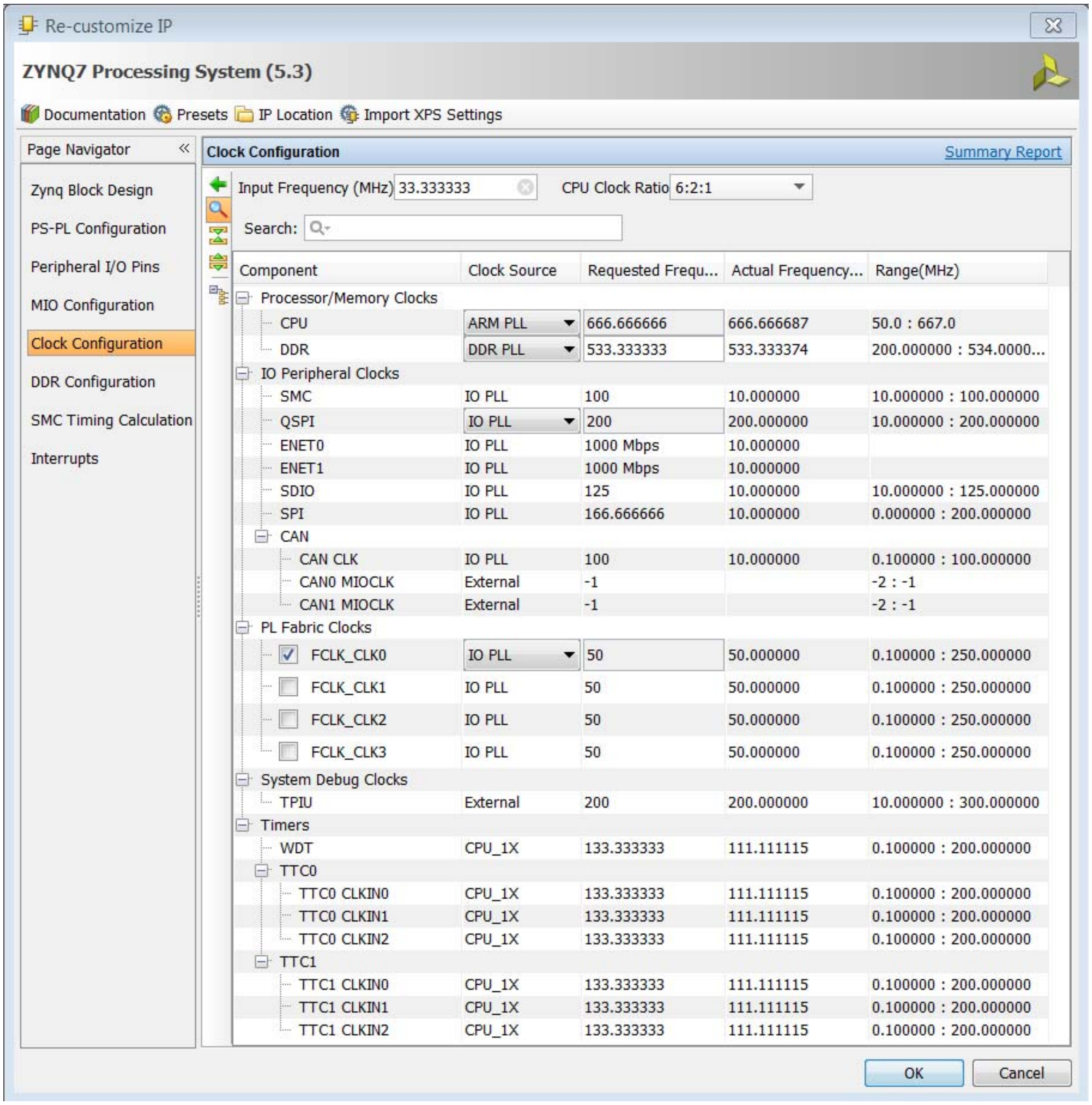
Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequ...	Actual Frequency...	Range(MHz)
⊞ Processor/Memory Clocks				
⊞ IO Peripheral Clocks				
⊞ PL Fabric Clocks				
⊞ System Debug Clocks				
⊞ Timers				

Figure 2-14: **Clock Configuration Page**

Figure 2-15 shows the Processor/Memory Clock configuration page.



Input Frequency (MHz) 33.333333

CPU Clock Ratio 6:2:1

Component	Clock Source	Requested Frequ...	Actual Frequency...	Range(MHz)
Processor/Memory Clocks				
CPU	ARM PLL	666.666666	666.666687	50.0 : 667.0
DDR	DDR PLL	533.333333	533.333374	200.000000 : 534.0000...
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	200.000000	10.000000 : 200.000000
ENET0	IO PLL	1000 Mbps	10.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	125	10.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000
CAN				
CAN CLK	IO PLL	100	10.000000	0.100000 : 100.000000
CAN0 MIOCLK	External	-1		-2 : -1
CAN1 MIOCLK	External	-1		-2 : -1
PL Fabric Clocks				
☑ FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK1	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 250.000000
System Debug Clocks				
TPIU	External	200	200.000000	10.000000 : 300.000000
Timers				
WDT	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0				
TTC0 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1				
TTC1 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000

Figure 2-15: **Processor and Memory Clock Configurations Page**

The *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 3] describes the clocking of the PS in detail. The Zynq clocking dialog box lets you set the peripheral clocks. The peripherals on the PS typically allow clock source selection from internal PLLs or from an external clock source. Most of the clocks can select the PLL to generate the clock.

Because the same PLL generates multiple frequencies, it might not be possible to get the exact frequency entered in the Requested Frequency (MHz) column. The achievable frequency is in the Actual Frequency (MHz) column.

Note: The frequency for a specific peripheral depends on many factors, such as input frequency, frequency for other peripherals driven from the same PLL, and restrictions from the architecture. Details of the M & D values chosen by the tool are available in the log file.

DDR Configuration

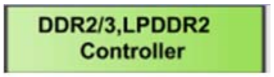


Figure 2-16: **DDR Controller**

You can configure DDR using one of two methods:

- From the Page Navigator, select the **DDR Configuration**.
- In the Zynq block design, click the **DDR2/3, LPDDR2** Controller block.

The DDR memory controller supports DDR2, DDR3, DDR3L, and LPDDR2 devices and consists of three major blocks: an AXI memory port interface - DDR interface (DDRI), a core controller with transaction scheduler (DDRC), and a controller with digital PHY (DDRP).

The DDRI block interfaces with four 64-bit synchronous AXI interfaces to serve multiple AXI masters simultaneously. Each AXI interface has a dedicated transaction FIFO. The DDRC contains two 32-entry content addressable memories (CAMs) to perform DDR data service scheduling to maximize DDR memory efficiency. It also contains a "fly-by" channel for a low-latency channel to allow access to DDR memory without going through the CAM.

The PHY processes read and write requests from the controller and translates them into specific signals within the timing constraints of the target DDR memory. The PHY uses signals from the controller to produce internal signals that connect to the pins using the digital PHYs. The DDR pins connect directly to the DDR device(s) using the PCB signal traces.

The system accesses the DDR using the DDRI through its four 64-bit AXI memory ports:

- One AXI port is dedicated to the L2-cache for the CPUs and ACP
- Two ports are dedicated to the AXI_HP interfaces
- The other masters on the AXI interconnect share the fourth port

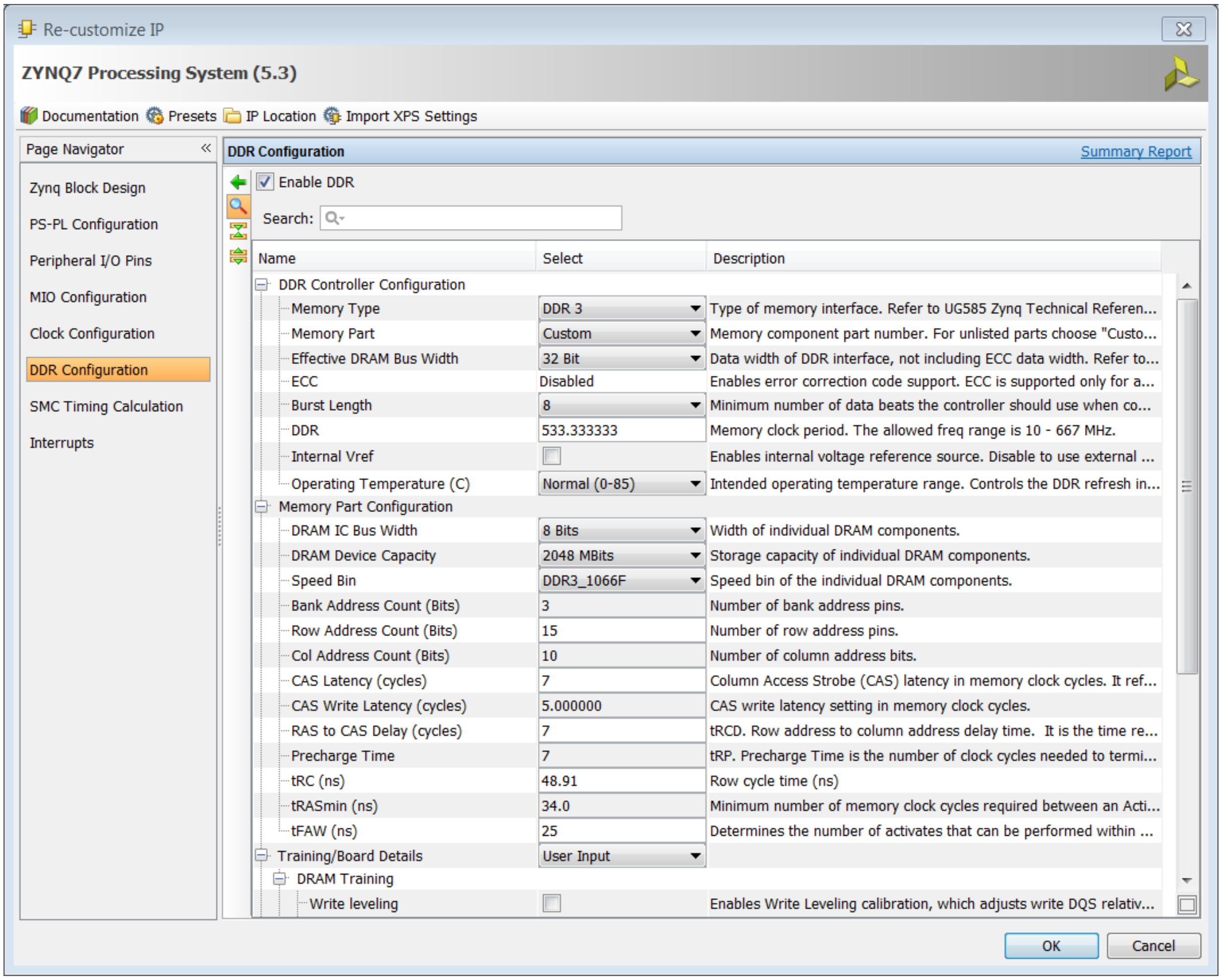
The DDRI arbitrates the requests from the eight ports (four reads and four writes). The arbiter selects a request and passes it to the DDR controller and transaction scheduler (DDRC).

The arbitration is based on a combination of how long the request has been waiting, the urgency of the request, and if the request is within the same page as the previous request.

The DDRC receives requests from the DDRI through a single interface for both read and write flows. Read requests include a tag field that the DDR returns with the data. The DDR controller PHY (DDRP) drives the DDR transactions.

Figure 2-17 shows the DDR Controller configuration page.

Note: 8-bit interfaces are not supported; however, 8-bit parts can be used to create 16/32-bit interfaces.



Re-customize IP

ZYNQ7 Processing System (5.3)

Documentation Presets IP Location Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

DDR Configuration — Summary Report

☑ Enable DDR

Search:

Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Type of memory interface. Refer to UG585 Zynq Technical Referen...
Memory Part	Custom	Memory component part number. For unlisted parts choose "Custo...
Effective DRAM Bus Width	32 Bit	Data width of DDR interface, not including ECC data width. Refer to...
ECC	Disabled	Enables error correction code support. ECC is supported only for a...
Burst Length	8	Minimum number of data beats the controller should use when co...
DDR	533.333333	Memory clock period. The allowed freq range is 10 - 667 MHz.
Internal Vref	□	Enables internal voltage reference source. Disable to use external ...
Operating Temperature (C)	Normal (0-85)	Intended operating temperature range. Controls the DDR refresh in...
Memory Part Configuration		
DRAM IC Bus Width	8 Bits	Width of individual DRAM components.
DRAM Device Capacity	2048 MBits	Storage capacity of individual DRAM components.
Speed Bin	DDR3_1066F	Speed bin of the individual DRAM components.
Bank Address Count (Bits)	3	Number of bank address pins.
Row Address Count (Bits)	15	Number of row address pins.
Col Address Count (Bits)	10	Number of column address bits.
CAS Latency (cycles)	7	Column Access Strobe (CAS) latency in memory clock cycles. It ref...
CAS Write Latency (cycles)	5.000000	CAS write latency setting in memory clock cycles.
RAS to CAS Delay (cycles)	7	tRCD. Row address to column address delay time. It is the time re...
Precharge Time	7	tRP. Precharge Time is the number of clock cycles needed to termi...
tRC (ns)	48.91	Row cycle time (ns)
tRASmin (ns)	34.0	Minimum number of memory clock cycles required between an Acti...
tFAW (ns)	25	Determines the number of activates that can be performed within ...
Training/Board Details	User Input	
DRAM Training		
Write leveling	□	Enables Write Leveling calibration, which adjusts write DQS relativ...

Figure 2-17: **DDR Controller Configurations Page**

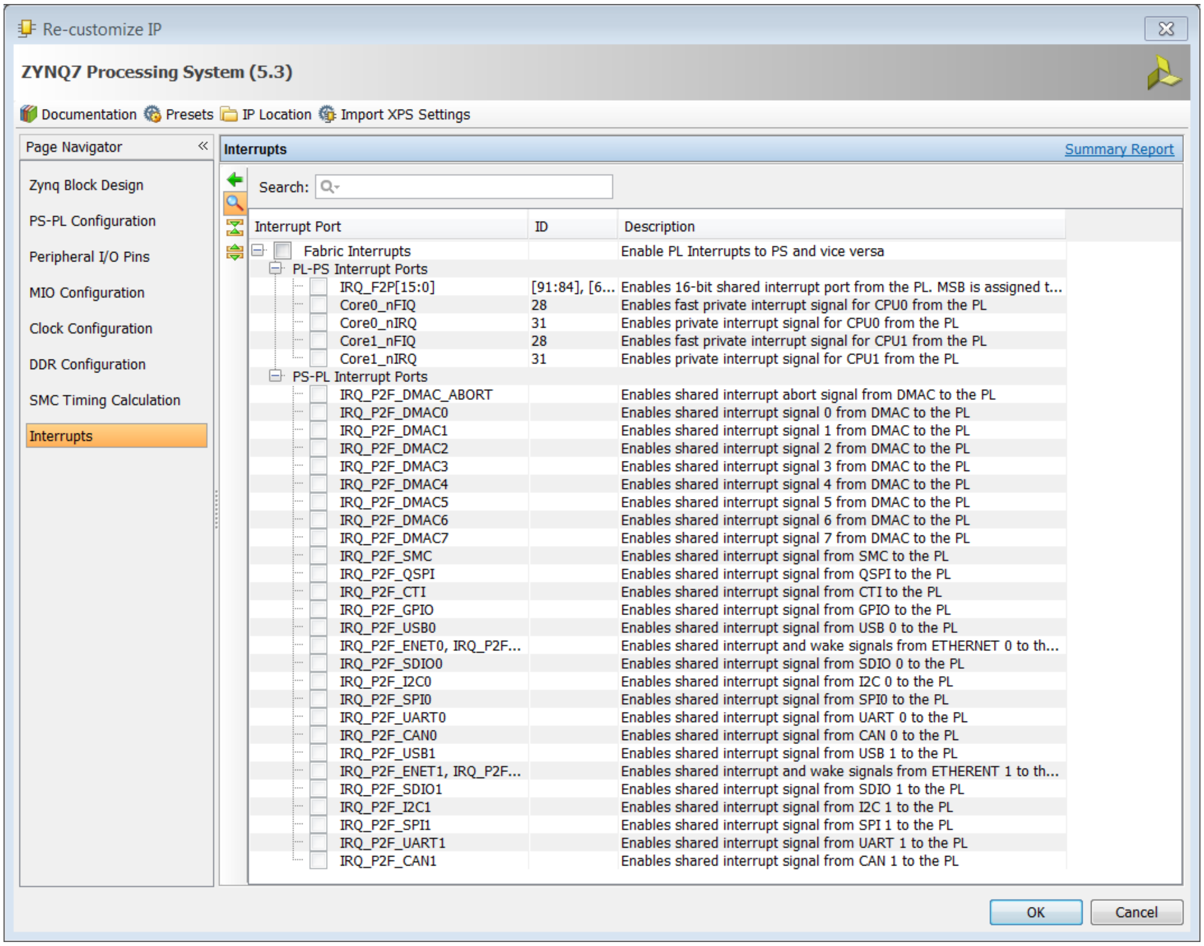
GIC - Interrupt Controller

You can configure the Generic Interrupt Controller (GIC) in one of two methods:

- In the Page Navigator, click **Interrupts**.
- In the Zynq block design, click the GIC block.

Figure 2-18: **Generic Interrupt Controller**

Figure 2-19 shows the Interrupt Port Configuration page.



Interrupts

Search:

Interrupt Port	ID	Description
⊟ □ Fabric Interrupts		Enable PL Interrupts to PS and vice versa
⊟ PL-PS Interrupt Ports		
□ IRQ_F2P[15:0]	[91:84], [6...	Enables 16-bit shared interrupt port from the PL. MSB is assigned t...
□ Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
□ Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
□ Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
□ Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
⊟ PS-PL Interrupt Ports		
□ IRQ_P2F_DMAC_ABORT		Enables shared interrupt abort signal from DMAC to the PL
□ IRQ_P2F_DMAC0		Enables shared interrupt signal 0 from DMAC to the PL
□ IRQ_P2F_DMAC1		Enables shared interrupt signal 1 from DMAC to the PL
□ IRQ_P2F_DMAC2		Enables shared interrupt signal 2 from DMAC to the PL
□ IRQ_P2F_DMAC3		Enables shared interrupt signal 3 from DMAC to the PL
□ IRQ_P2F_DMAC4		Enables shared interrupt signal 4 from DMAC to the PL
□ IRQ_P2F_DMAC5		Enables shared interrupt signal 5 from DMAC to the PL
□ IRQ_P2F_DMAC6		Enables shared interrupt signal 6 from DMAC to the PL
□ IRQ_P2F_DMAC7		Enables shared interrupt signal 7 from DMAC to the PL
□ IRQ_P2F_SMC		Enables shared interrupt signal from SMC to the PL
□ IRQ_P2F_QSPI		Enables shared interrupt signal from QSPI to the PL
□ IRQ_P2F_CTI		Enables shared interrupt signal from CTI to the PL
□ IRQ_P2F_GPIO		Enables shared interrupt signal from GPIO to the PL
□ IRQ_P2F_USB0		Enables shared interrupt signal from USB 0 to the PL
□ IRQ_P2F_ENET0, IRQ_P2F...		Enables shared interrupt and wake signals from ETHERNET 0 to th...
□ IRQ_P2F_SDIO0		Enables shared interrupt signal from SDIO 0 to the PL
□ IRQ_P2F_I2C0		Enables shared interrupt signal from I2C 0 to the PL
□ IRQ_P2F_SPI0		Enables shared interrupt signal from SPI0 to the PL
□ IRQ_P2F_UART0		Enables shared interrupt signal from UART 0 to the PL
□ IRQ_P2F_CAN0		Enables shared interrupt signal from CAN 0 to the PL
□ IRQ_P2F_USB1		Enables shared interrupt signal from USB 1 to the PL
□ IRQ_P2F_ENET1, IRQ_P2F...		Enables shared interrupt and wake signals from ETHERENT 1 to th...
□ IRQ_P2F_SDIO1		Enables shared interrupt signal from SDIO 1 to the PL
□ IRQ_P2F_I2C1		Enables shared interrupt signal from I2C 1 to the PL
□ IRQ_P2F_SPI1		Enables shared interrupt signal from SPI 1 to the PL
□ IRQ_P2F_UART1		Enables shared interrupt signal from UART 1 to the PL
□ IRQ_P2F_CAN1		Enables shared interrupt signal from CAN 1 to the PL

Figure 2-19: **GIC Interrupts**

GIC is a centralized resource for managing interrupts sent to the CPUs from the PS and PL. The controller enables, disables, masks, and prioritizes the interrupt sources and sends them to the selected CPU (or CPUs) in a programmed manner as the CPU interface accepts the next interrupt. In addition, the controller supports security extension for implementing a security-aware system.

The controller is based on the ARM Generic Interrupt Controller Architecture version 1.0 (GIC v1), non-vectored.

The private bus on the CPU accesses the registers for fast read/write response by avoiding temporary blockage or other bottlenecks in the Interconnect.

The interrupt distributor centralizes all interrupt sources before dispatching the one with the highest priority to the individual CPUs.

The GIC ensures that, when you target an interrupt to several CPUs, only one CPU takes the interrupt at a time. All interrupt sources contain a unique interrupt ID number. All interrupt sources have their own configurable priority and list of targeted CPUs.

Both the *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 3] and the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 1] contain information regarding the logic blocks in the Zynq-7000 device.

Interconnect between PS and PL

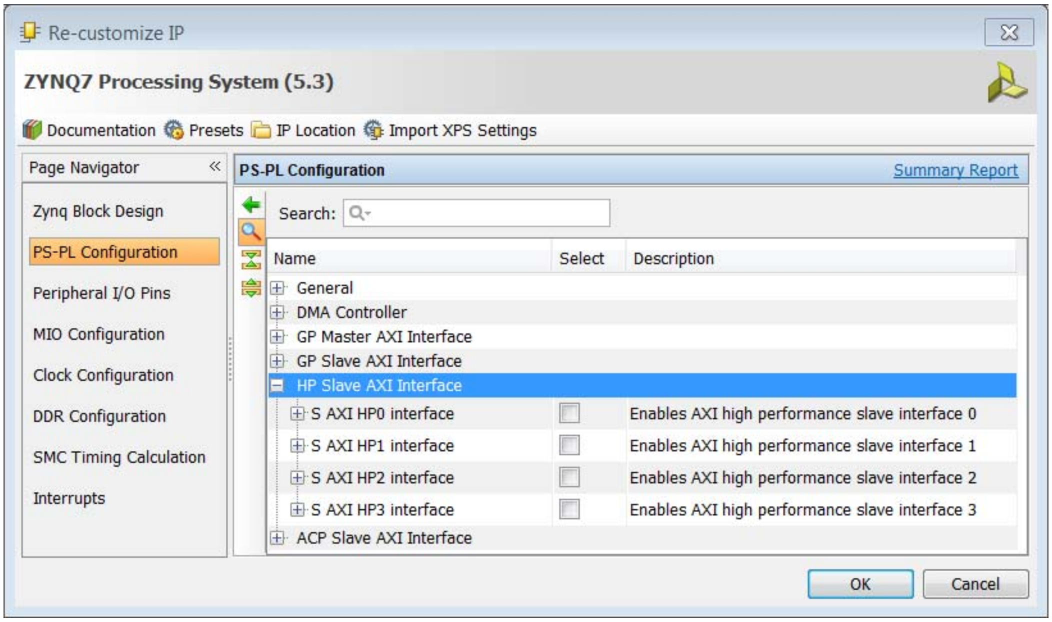
AXI_HP Interfaces

Figure 2-20: **AXI_HP Interfaces**

The four AXI_HP interfaces provide PL bus masters with high-bandwidth data paths to the DDR and OCM memories. Each interface includes two FIFO buffers for read and write traffic. The PL to the memory Interconnect routes the high-speed AXI_HP ports either to two DDR memory ports or to the OCM. The AXI_HP interfaces are also referenced as AXI FIFO interfaces (AFI), to emphasize their buffering capabilities.

IMPORTANT: *You must enable the PL level shifters using LVL_SHFTR_EN before PL logic communication can occur.*

Enable these interfaces by selecting **PS-PL Configuration** from the Page Navigator and expanding the **HP Slave AXI Interface** option as shown in Figure 2-21.



Name	Select	Description
General		
DMA Controller		
GP Master AXI Interface		
GP Slave AXI Interface		
HP Slave AXI Interface		
S AXI HP0 interface	□	Enables AXI high performance slave interface 0
S AXI HP1 interface	□	Enables AXI high performance slave interface 1
S AXI HP2 interface	□	Enables AXI high performance slave interface 2
S AXI HP3 interface	□	Enables AXI high performance slave interface 3
ACP Slave AXI Interface		

Figure 2-21: **Enabling AXI HP Interfaces**

The interfaces provide a high-throughput data path between PL masters and PS memories including the DDR and on-chip RAM. The main features include:

32- or 64-bit data wide master interfaces (independently programmed per port)

- Efficient dynamic upsizing to 64 bits for aligned transfers in 32-bit interface mode, controllable using `AxCACHE`
- Automatic expansion to 64-bits for unaligned 32-bit transfers in 32-bit interface mode
- Programmable release threshold of write commands
- Asynchronous clock frequency domain crossing for all AXI interfaces between the PL and PS
- Smoothing out of "long-latency" transfers using 1 KB (128 by 64 bit) data FIFOs for both reads and writes
- QoS signaling available from PL ports
- Command and Data FIFO fill-level counts available to the PL
- Standard AXI 3.0 interfaces support
- Programmable command issuance to the interconnect, separately for read and write commands
- Large slave interface read acceptance capability in the range of 14 to 70 commands (burst length dependent)
- Large slave interface write acceptance capability in the range of 8 to 32 commands (burst length dependent)

AXI_ACP Interface

The Accelerator Coherency Port (ACP) provides low-latency access to programmable logic masters, with optional coherency and L1 and L2 cache.

From a system perspective, the ACP interface has similar connectivity as the APU CPUs. Due to this close connectivity, the ACP directly competes for resource access outside of the APU block.

IMPORTANT: *You must enable the PL level shifters using LVL_SHFTR_EN before PL logic communication can occur.*

In the ZYNQ7 block design, click the **64b AXI ACP Slave Ports** block to configure the AXI_ACP.

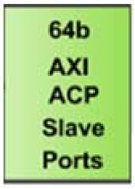
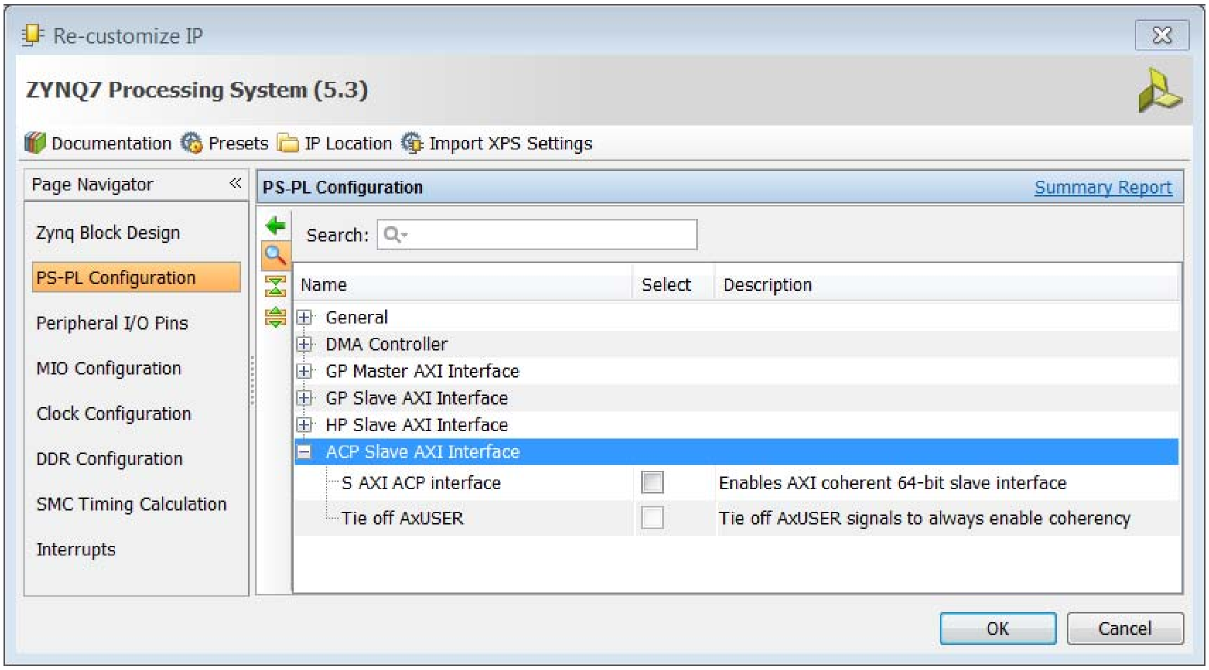


Figure 2-22: **AXI_ACP Configuration**

Alternatively, select the **PS-PL Configuration** and expand **ACP Slave AXI Interface**.

Figure 2-23 shows the ACP AXI Slave Configuration page.


Re-customize IP

ZYNQ7 Processing System (5.3)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator

- Zynq Block Design
- PS-PL Configuration
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts

PS-PL Configuration | Summary Report

Search:

Name	Select	Description
⊞ General		
⊞ DMA Controller		
⊞ GP Master AXI Interface		
⊞ GP Slave AXI Interface		
⊞ HP Slave AXI Interface		
⊟ ACP Slave AXI Interface		
S AXI ACP interface	□	Enables AXI coherent 64-bit slave interface
Tie off AxUSER	□	Tie off AxUSER signals to always enable coherency

OK | Cancel

Figure 2-23: **ACP Slave AXI Interface Page**

AXI_GP Interfaces

AXI_GP features include:

- Standard AXI protocol
- Data bus width: 32
- Master port ID width: 12
- Master port issuing capability: 8 reads, 8 writes
- Slave port ID width: 6
- Slave port acceptance capability: 8 reads, 8 writes

These interfaces are connected directly to the ports of the master interconnect and the slave interconnect without additional FIFO buffering, unlike the AXI_HP interfaces, which have elaborate FIFO buffering to increase performance and throughput. Therefore, the performance is constrained by the ports of the master interconnect and the slave interconnect. These interfaces are for general-purpose use only; they are not intended to achieve high performance.

IMPORTANT: *You must enable the PL level shifters using LVL_SHFTR_EN before PL logic communication can occur.*

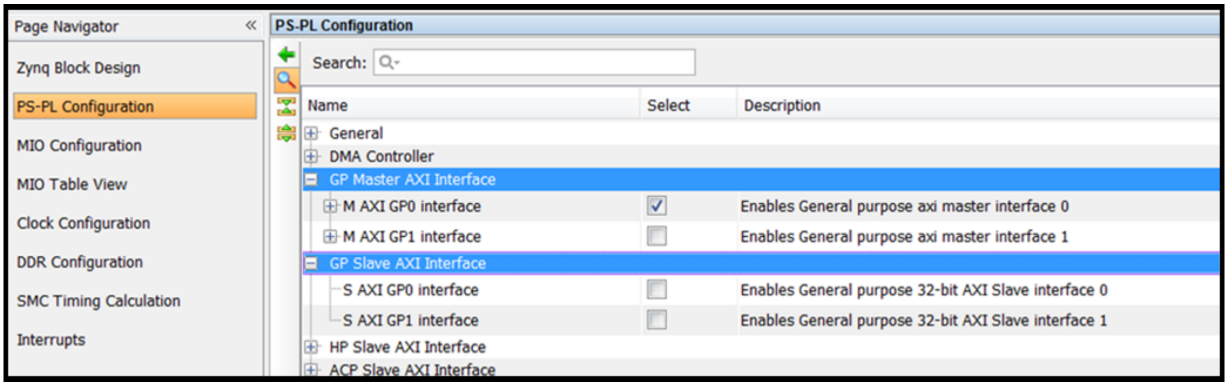
In the ZYNQ7 block design, click the following block to configure the AXI_GP interface.

32b GP AXI Master Ports
32b GP AXI Slave Ports

Figure 2-24: **AXI_GP Configuration**

Alternatively, in the Page Navigator, select the **PS-PL Configuration** and expand the **GP Master AXI Interface** and **GP Slave AXI Interface** options.

Figure 2-25 shows the GP AXI Master and Slave Configuration page.



Name	Select	Description
General		
DMA Controller		
GP Master AXI Interface		
M AXI GP0 interface	☑	Enables General purpose axi master interface 0
M AXI GP1 interface	□	Enables General purpose axi master interface 1
GP Slave AXI Interface		
S AXI GP0 interface	□	Enables General purpose 32-bit AXI Slave interface 0
S AXI GP1 interface	□	Enables General purpose 32-bit AXI Slave interface 1
HP Slave AXI Interface		
ACP Slave AXI Interface		

Figure 2-25: **GP Master and Slave AXI Interfaces**

Using the Programmable Logic (PL)

The PL provides a rich architecture of user-configurable capabilities.

Configurable logic blocks (CLB)

- 6-input look-up tables (LUTs) with memory capability within the LUT
- Register and shift register functionality
- Adders that can be cascaded

36 Kb block RAM

- Dual ports, up to 72 bits wide
- Configurable as dual 18 Kb
- Programmable FIFO logic
- Built-in error correction circuitry

Digital signal processing - DSP48E1 Slice

- 25 × 18 two's complement multiplier/accumulator high-resolution (48 bit) signal processor
- Power-saving 25-bit pre-adder to optimize symmetrical filter applications
- Advanced features: optional pipelining, optional ALU, and dedicated buses for cascading

Clock management

- UHigh-speed buffers and routing for low-skew clock distribution
- Frequency synthesis and phase shifting
- Low-jitter clock generation and jitter filtering

Configurable I/Os

- High-performance SelectIO™ technology
- High-frequency decoupling capacitors within the package for enhanced signal integrity
- Digitally controlled impedance that can be tri-state for lowest power, high-speed I/O operation
- High range (HR) I/Os support 1.2 V to 3.3 V
- High performance (HP) I/Os support 1.2 V to 1.8 V (7z030, 7z045, and 7z100 devices)

Low-power gigabit transceivers

- (7z030, 7z045, and 7z100 devices)
- High-performance transceivers capable of up to 12.5 Gb/s (GTX)
- Low-power mode optimized for chip-to-chip interfaces
- Advanced transmit pre- and post-emphasis, and receiver linear (CTLE) and decision feedback equalization (DFE), including adaptive equalization for additional margin

Analog-to-digital converter (XADC)

- Dual 12-bit 1 MSPS analog-to-digital converters (ADCs)
- Up to 17 flexible and user-configurable analog inputs
- On-chip or external reference option
- On-chip temperature (±4°C max error) and power supply (±1% max error) sensors
- Continuous JTAG access to ADC measurements

Integrated interface blocks for PCI Express designs (7z030, 7z045, and 7z100 devices)

- Compatible to the PCI Express base specification 2.1 with Endpoint and Root Port capability
- Supports Gen1 (2.5 Gb/s) and Gen2 (5.0 Gb/s) speeds

Advanced configuration options, advanced error reporting (AER), end-to-end CRC (ECRC)

Custom Logic

The Vivado® IP packager lets you and third-party IP developers use the Vivado IDE to easily prepare an Intellectual Property (IP) design for use in the Vivado IP catalog. The IP user can then instantiate this third party IP into a design in the Vivado Design Suite.

When IP developers use the Vivado Design Suite IP packaging flow, the IP user has a consistent experience whether using Xilinx IP, third-party IP, or customer-developed IP within the Vivado Design Suite.

IP developers can use the IP packager feature to package IP files and associated data into a ZIP file. The IP user receives this generated ZIP file and installs the IP into the Vivado Design Suite IP Catalog. The IP user then customizes the IP through parameter selections and generates an instance of the IP. See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 2] and *Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator* (UG995) [Ref 4] for more information.

RECOMMENDED: *To verify the proper packaging of the IP before handing it off to the IP user, Xilinx® recommends that the IP developer run each IP module completely through the IP user flow to validate that the IP is ready for use.*

Zynq-7000 Processing System Simulation

The Zynq®-7000 Bus Functional Model (BFM) supports the functional simulation of Zynq-7000-based applications. It enables the functional verification of Programmable Logic (PL) by mimicking the PS-PL interfaces and OCM/DDR memories of Processor System (PS) logic. This BFM is delivered as a package of encrypted Verilog modules. A sequence of Verilog tasks in a Verilog syntax file controls the BFM operation.

Features

- Pin compatible and Verilog-based simulation model
- Supports all AXI interfaces
    - AXI 3.0 compliant
- Sparse memory model (for DDR) and a RAM model (for OCM)
- Verilog task-based API
- Delivered in Vivado Design Suite
- Blocking and non-blocking interrupt support
- Requires license to AXI BFM models

Applications

The Zynq-7000 BFM provides a simulation environment for the Zynq-7000 PS logic, typically replacing the processing_system7 block in a design. The Zynq-7000 BFM models the following:

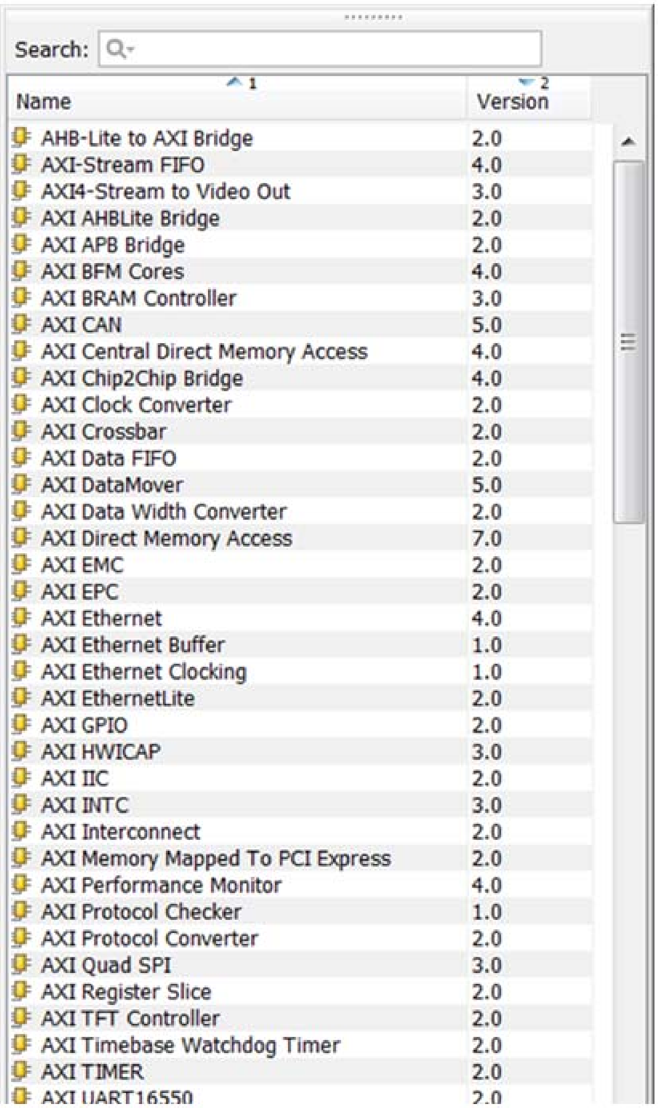
- Transactions originating from PS masters through the AXI BFM master API calls
- Transactions terminating through the PS slaves to models of the OCM and DDR memories through interconnect models
- FCLK reset and clocking support
- Input interrupts to the PS from PL
- PS register map

For more information on the Zynq BFM, see *Zynq-7000 Bus Functional Model* (DS897).

Embedded IP Catalog

The Vivado Design Suite IP Catalog is a unified repository that lets you search, review detailed information, and view associated documentation for the IP. After you add the third-party or customer IP to the Vivado Design Suite IP catalog, you can access the IP through the Vivado Design Suite flows.

Figure 2-26 shows a portion of the Vivado IDE IP integrator IP Catalog.



Name	Version
AHB-Lite to AXI Bridge	2.0
AXI-Stream FIFO	4.0
AXI4-Stream to Video Out	3.0
AXI AHBLite Bridge	2.0
AXI APB Bridge	2.0
AXI BFM Cores	4.0
AXI BRAM Controller	3.0
AXI CAN	5.0
AXI Central Direct Memory Access	4.0
AXI Chip2Chip Bridge	4.0
AXI Clock Converter	2.0
AXI Crossbar	2.0
AXI Data FIFO	2.0
AXI DataMover	5.0
AXI Data Width Converter	2.0
AXI Direct Memory Access	7.0
AXI EMC	2.0
AXI EPC	2.0
AXI Ethernet	4.0
AXI Ethernet Buffer	1.0
AXI Ethernet Clocking	1.0
AXI EthernetLite	2.0
AXI GPIO	2.0
AXI HWICAP	3.0
AXI IIC	2.0
AXI INTC	3.0
AXI Interconnect	2.0
AXI Memory Mapped To PCI Express	2.0
AXI Performance Monitor	4.0
AXI Protocol Checker	1.0
AXI Protocol Converter	2.0
AXI Quad SPI	3.0
AXI Register Slice	2.0
AXI TFT Controller	2.0
AXI Timebase Watchdog Timer	2.0
AXI TIMER	2.0
AXI UART16550	2.0

Figure 2-26: **IP Integrator IP Catalog**

Completing Connections

After you have configured the ZYNQ-7000 PS, you can instantiate other IP that go in the programmable logic portion of the device.

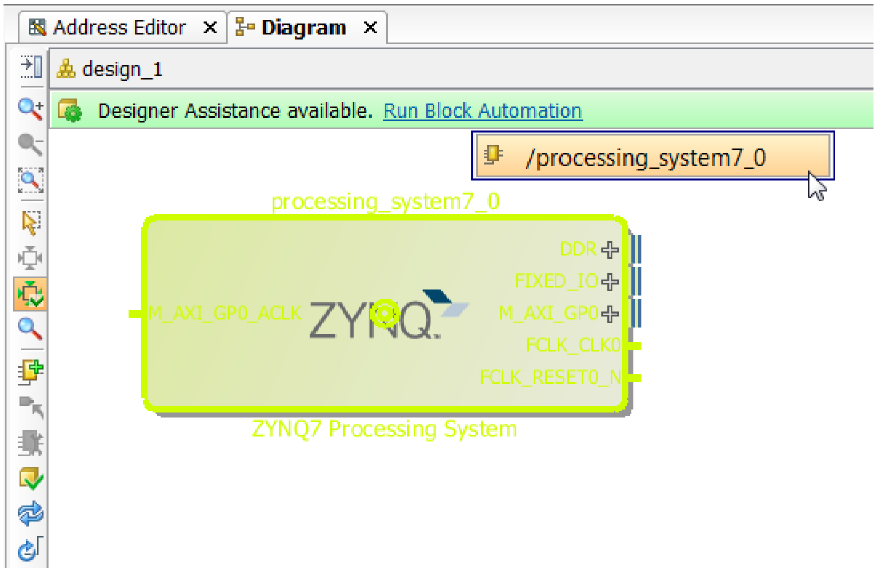
In the IP integrator diagram area, right-click and select **Add IP**.

You can use two built-in features of the IP integrator to complete the rest of the IP subsystem design: Block Automation and Connection Automation. These features help you put together a basic microprocessor system in the IP integrator tool and connect ports to external I/O ports.

Block Automation

Block Automation is available when a microprocessor such as the Zynq-7000 PS MicroBlaze™ processor is instantiated in the block design of the IP integrator tool.

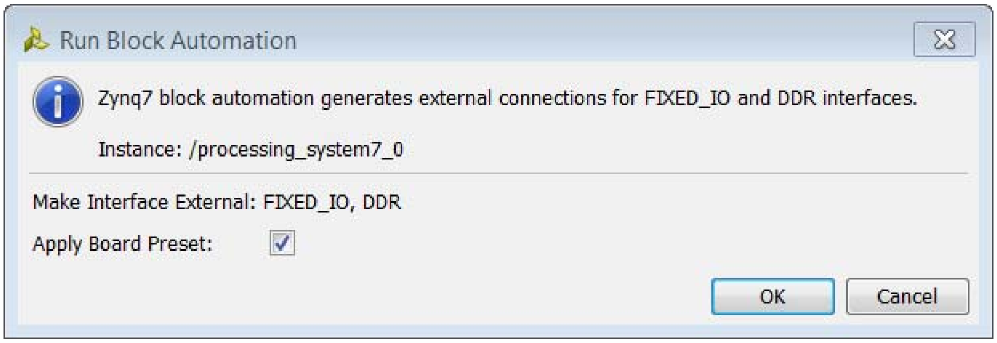
Click **Run Block Automation** to get assistance with putting together a simple **ZYNQ Processing System**, as shown in Figure 2-27.



Designer Assistance available. Run Block Automation

Figure 2-27: **Run Block Automation Feature**

The Run Block Automation dialog box shows the options available for automation, as shown in Figure 2-28.



Zynq7 block automation generates external connections for FIXED_IO and DDR interfaces.

Instance: /processing_system7_0

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ☑

Figure 2-28: **Run Block Automation for Zynq Processor Dialog Box**

After you click **OK**, the Block Automation feature creates the basic system, as shown in Figure 2-29.

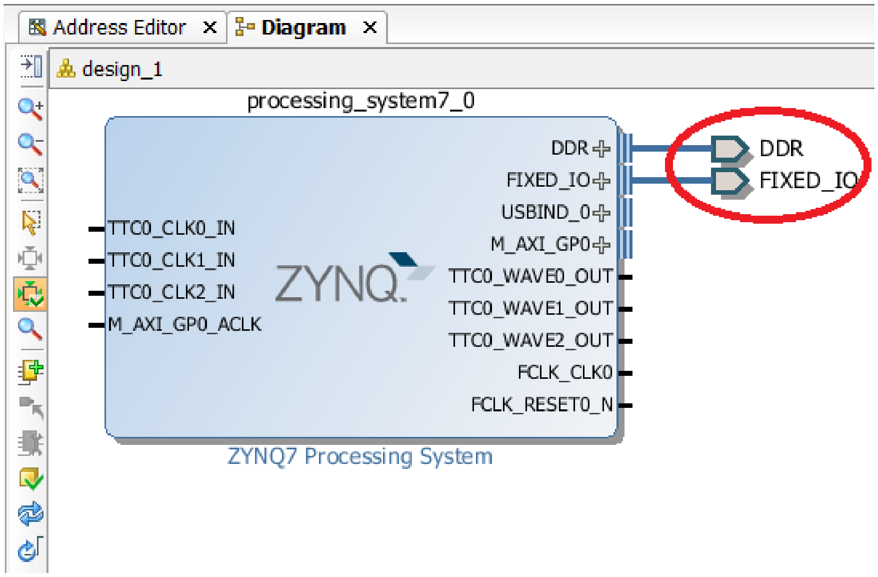


Figure 2-29: **IP Integrator Window after Running Block Automation**

The Vivado IP integrator tool also provides a Board Automation feature when using a Xilinx Target Reference Platform, such as the ZC702.

This feature provides connectivity of the ports of an IP to the FPGA pins on the target board. The IP configures accordingly, and based on your selections, connects the I/O ports. Board Automation automatically generates the physical constraints for those IP that require physical constraints.

In Figure 2-29, observe that the external `DDR` and `FIXED_IO` interfaces connect to external ports.

Using Connection Automation

If the IP integrator tool determines that a potential connection exists among the instantiated IP in the canvas, it opens the Connection Automation feature.

In Figure 2-30, the AXI BRAM Controller and the Block Memory Generator IP are instantiated along with the ZYNQ7 Processing System IP.

The IP integrator tool determines that a potential connection exists between the AXI BRAM Controller and the ZYNQ7 IP; consequently, Connection Automation is available.

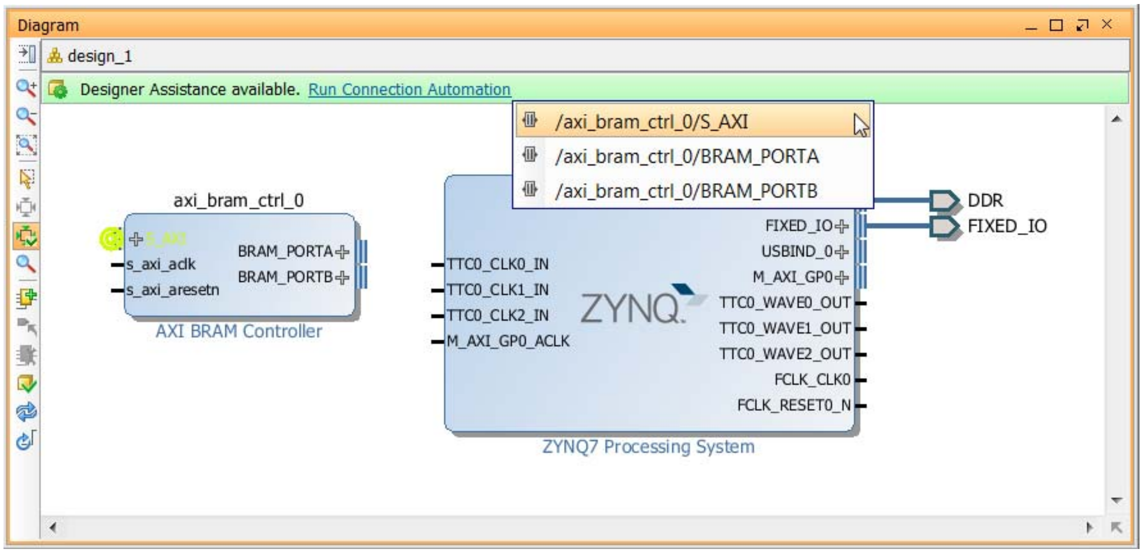

Designer Assistance available. Run Connection Automation

Figure 2-30: **Connection Automation Feature**

Clicking **Run Connection Automation** does the following:

- Instantiates an AXI Interconnect, and a Proc Sys Reset IP
- Connects the AXI BRAM Controller to the ZYNQ7 PS IP using the AXI Interconnect
- Appropriately connects the Proc Sys Reset IP as shown in Figure 2-31

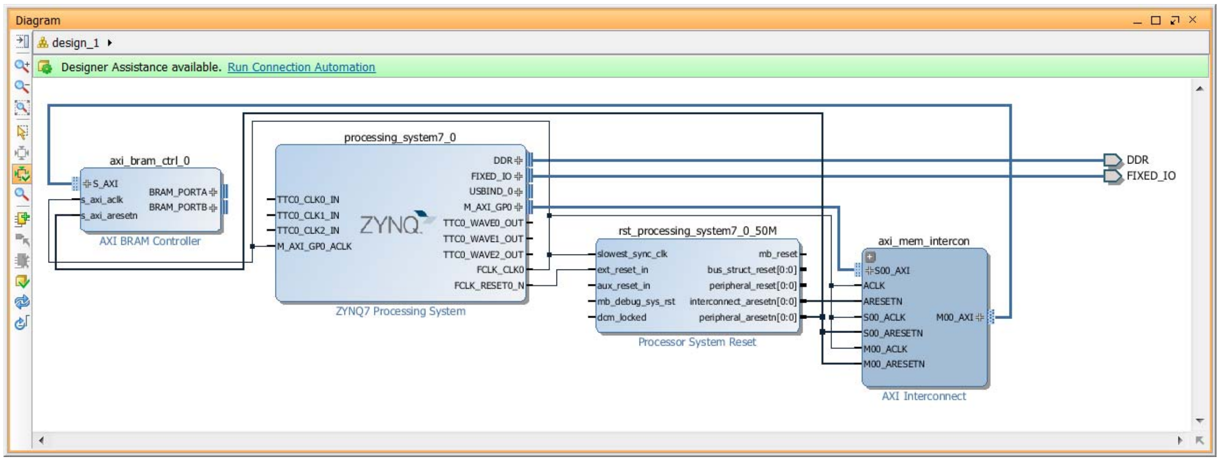
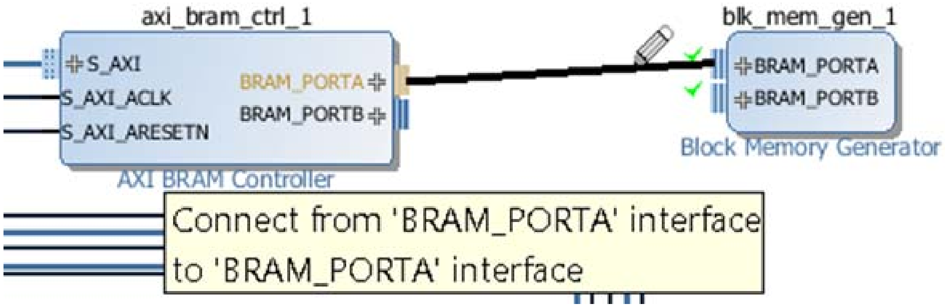

Designer Assistance available. Run Connection Automation

Figure 2-31: **Block Design after Connection Automation**

Manual Connections in a Design

Figure 2-32 shows that you need to connect the AXI BRAM Controller to the Block Memory Generator. You can do this manually.

1. As you move the cursor near an interface or pin connector on an IP block, the cursor changes to a pencil.
2. Click an interface or pin connector on an IP block, and drag the connection to the destination block.



Connect from 'BRAM_PORTA' interface to 'BRAM_PORTA' interface

Figure 2-32: **Manually Connecting Ports**

Manually Creating and Connecting to I/O Ports

You can manually create external I/O ports in the Vivado IP integrator. You can connect signals or interfaces to external I/O ports by selecting a pin, a bus, or an interface connection.

To manually create/connect to an I/O port, right-click the port in the block diagram, and then select one of the following from the right-click menu:

- **Make External**. You can use the **Ctrl+Click** keyboard combination to select multiple pins and invoke the **Make External** connection. This command ties a pin on an IP to an I/O port on the block design.
- **Create Port**. Use this command for non-interface signals, such as a `clock`, `reset`, or `uart_txd`.

  The Create Port option gives more control in terms of specifying the input/output, the bit-width and the type (`clk`, `reset`, or `data`). In case of a clock, you can even specify the input frequency.
- **Create Interface Port**. This command creates ports on the interface for groupings of signals that share a common function.

  For example, the S_AXI is an interface port on several Xilinx IP. The command gives more control in terms of specifying the interface type and the mode (master or slave).

Memory Mapping in Address Editor

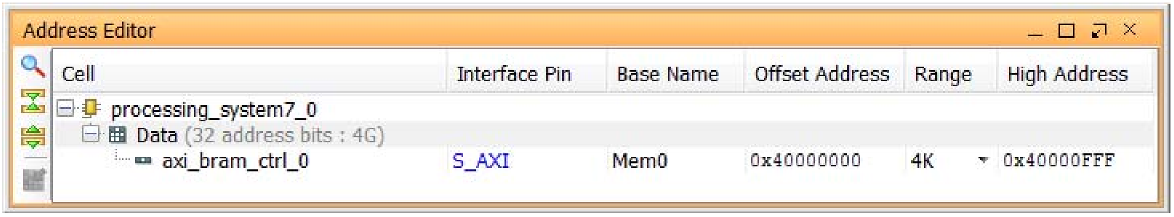
To generate the address map for this design:

1. Click the Address Editor tab above the diagram.
2. Click the **Auto Assign Address** button (bottom on the left side).

If you generate the RTL from an IP integrator diagram without first generating addresses, a prompt opens that lets you select to have the tool automatically assign addresses.

You can manually set addresses by entering values in **Offset Address** and **Range** columns.

Note: The Address Editor tab only opens if the diagram contains an IP block that functions as a bus master (such as the ZYNQ7 processor) in the design.



Cell	Interface Pin	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0x40000000	4K	0x40000FFF

Figure 2-33: **Memory Mapping Peripherals**

Running Design Rule Checks

The Vivado IP integrator runs basic DRCs in real time as you put the design together. However, errors can occur during design creation. For example, the frequency on a clock pin might not be set correctly.

To run a comprehensive DRC, click the **Validate Design** toolbar button ✓ .

If no warnings or errors occur in the design, a validation dialog box displays to confirm that there are no errors or critical warnings in your design,

Integrating a Block Design in the Top-Level Design

After you complete the block design and validate the design, there are two more steps required to complete the design:

- Generate the output products
- Create a HDL wrapper

Generating output products makes the source files and the appropriate constraints for the IP available in the Vivado IDE Sources window.

Depending upon what you selected as the target language during project creation, the IP integrator tool generates the appropriate files. If the Vivado IDE cannot generate the source files for a particular IP in the specified target language, a message displays in the console.

Generating Output Products

To generate output products, do one of the following:

- In the Block Design panel, expand the Design Sources hierarchy and select **Generate Output Products**.
- In the Flow Navigator panel, under IP Integrator, click **Generate Block Design**.

You can integrate an IP integrator block design into a higher-level design. To do so, instantiate the design in a higher-level HDL file.

Creating an HDL Wrapper

To instantiate at a higher level, in the Design Sources hierarchy of the Block Design panel, right-click the design and select **Create HDL Wrapper**.

This generates a top-level HDL file for the IP integrator subsystem. You can now take your design through the other design flows: elaboration, synthesis, and implementation.

Vivado Pin Planner View of PS I/O

The *Zynq-7000 All Programmable SoC PCB Design and Pin Planning Guide* (UG933) [Ref 5] provides a detailed description of guidelines for PCB Design and Pin Planning for Zynq-7000 devices.

Vivado IDE Generated Embedded Files

When you export a Zynq-7000 processor hardware design from the IP integrator tool to SDK, the IP integrator generates the following files:

Table 2-1: **Files Generated by IP Integrator**

File	Description
`system.xml`	This file opens by default when you launch SDK and displays the address map of your system.
`ps7_init.c` `ps7_init.h`	The ps7_init.c and ps7_init.h files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, PLLs, and MIOs. SDK uses these settings when initializing the processing system so applications can run on top of the processing system. Some settings in the processing system are in a fixed state for the ZC702 evaluation board.
`ps7_init.tcl`	This is the Tcl version of the INIT file.
`ps7_init.html`	The INIT file describes the initialization data.

See the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 1] for more information about generated files.

Using the Software Development Kit (SDK)

The Xilinx Software Development Kit (SDK) provides a complete environment for creating software applications targeted for Xilinx embedded processors. It includes a GNU-based compiler toolchain (GCC compiler, GDB debugger, utilities, and libraries), JTAG debugger, flash programmer, drivers for Xilinx IP and bare-metal board support packages, middleware libraries for application-specific functions, and an IDE for C/C++ bare-metal and Linux application development and debugging. Based upon the open source Eclipse platform, SDK incorporates the C/C++ Development Toolkit (CDT).

Features include:

- C/C++ code editor and compilation environment
- Project management
- Application build configuration and automatic makefile generation
- Error navigation
- Integrated environment for debugging and profiling embedded targets
- Additional functionality available using third-party plug-ins, including source code version control

SDK Availability

SDK is available from the Xilinx Vivado Design Suite installation package or as a standalone installation. SDK also includes an application template for creating a First Stage Bootloader (FSBL), as well as a graphical interface for building a boot image. SDK contains a help system that describes concepts, tasks, and reference information.

Exporting a Hardware Description

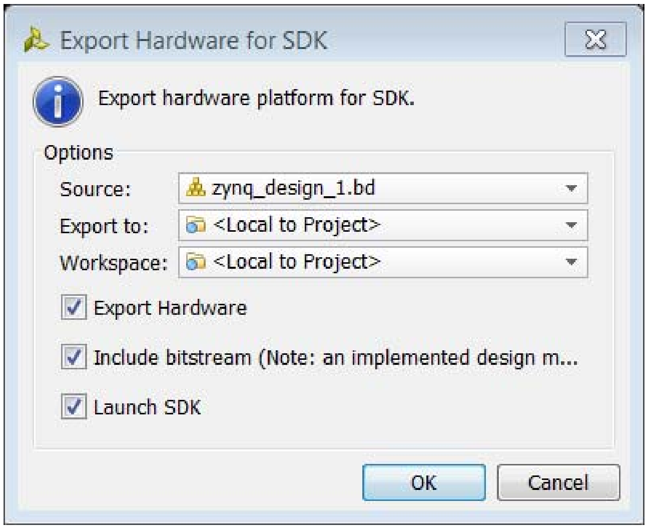
In the Flow Navigator, under IP Integrator, click **Open Block** to invoke the IP integrator design.

Now you are ready to export your design to SDK.

1. In the main Vivado IDE, select **File > Export Hardware for SDK**.

   The Export Hardware for SDK dialog box opens.

2. Ensure that you have checked the **Export Hardware**, **Include Bitstream**, and **Launch SDK** check boxes, as shown in Figure 2-34.



Source: zynq_design_1.bd

Export to: <Local to Project>

Workspace: <Local to Project>

☑ Export Hardware

☑ Include bitstream (Note: an implemented design m...

☑ Launch SDK

Figure 2-34: **Export Hardware for SDK**

Note: You must open the implemented design before you can select the **Include bitstream** check box.

After you export the hardware definition to SDK, and launch SDK, you can start writing your software application in SDK.

You can do further debug and downloading of the software from SDK.

Alternatively, you can import the ELF file for the software back into the Vivado tools, and integrate it with the FPGA bitstream for further download and testing.

Chapter 3

Using a MicroBlaze Processor in an Embedded Design

Introduction to MicroBlaze Processor Design

The Vivado IDE IP integrator is a powerful tool that lets you stitch together a processor-based system.

The MicroBlaze™ embedded processor is a Reduced Instruction Set Computer (RISC) core, optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs).

Figure 3-1 shows a functional block design of the MicroBlaze core.

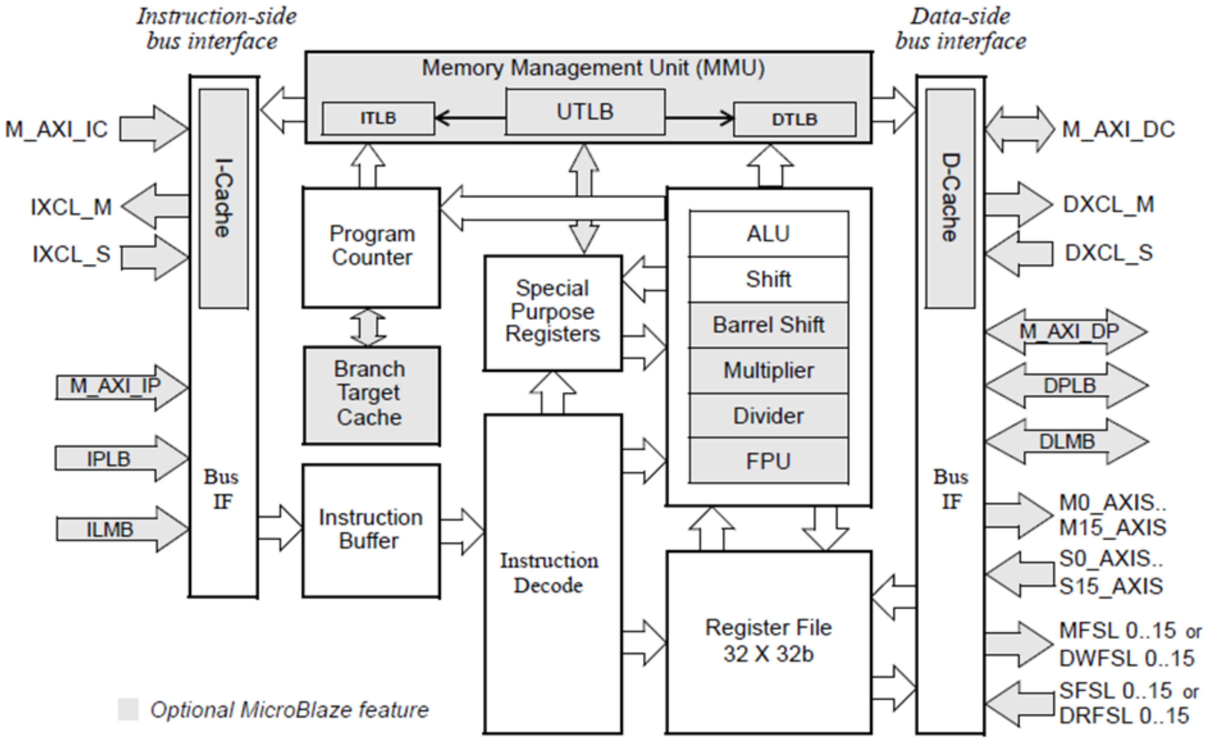


Figure 3-1: **Block Design of MicroBlaze Core**

The MicroBlaze processor is highly configurable: you can select a specific set of features required by your design.

The fixed feature set of the processor includes:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue pipeline

In addition to these fixed features, the MicroBlaze processor has parameterized values that allow selective enabling of additional functionality.

✓ **RECOMMENDED:** *Older (deprecated) versions of MicroBlaze support a subset of the optional features described in this manual. Only the latest (preferred) version of MicroBlaze (v9.0) supports all options. Xilinx recommends that new designs use the latest preferred version of the MicroBlaze processor.*

Refer to the *MicroBlaze Processor Reference Guide* (UG081) [Ref 6] for more information about the MicroBlaze processor design.

Creating an IP Integrator Design with the MicroBlaze Processor

Designing with a MicroBlaze processor is different using the Vivado IDE than it was using the ISE® Design Suite and Embedded Development Kit (EDK).

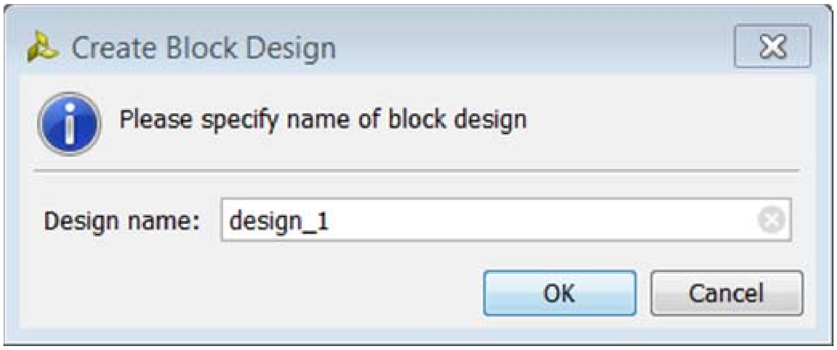
The Vivado IDE uses the IP integrator tool for embedded development. The IP integrator is a GUI-based interface that lets you stitch together complex IP subsystems.

A variety of IP are available in the Vivado IDE IP Catalog to meet the needs of complex designs.

You can also add custom IP to the IP Catalog.

Creating an IP Integrator Design with the MicroBlaze Processor

In the Flow navigator panel, under IP Integrator, click the **Create Block Design** button to open the Create Block Design dialog box. Enter the Design Name, as shown in Figure 3-2.



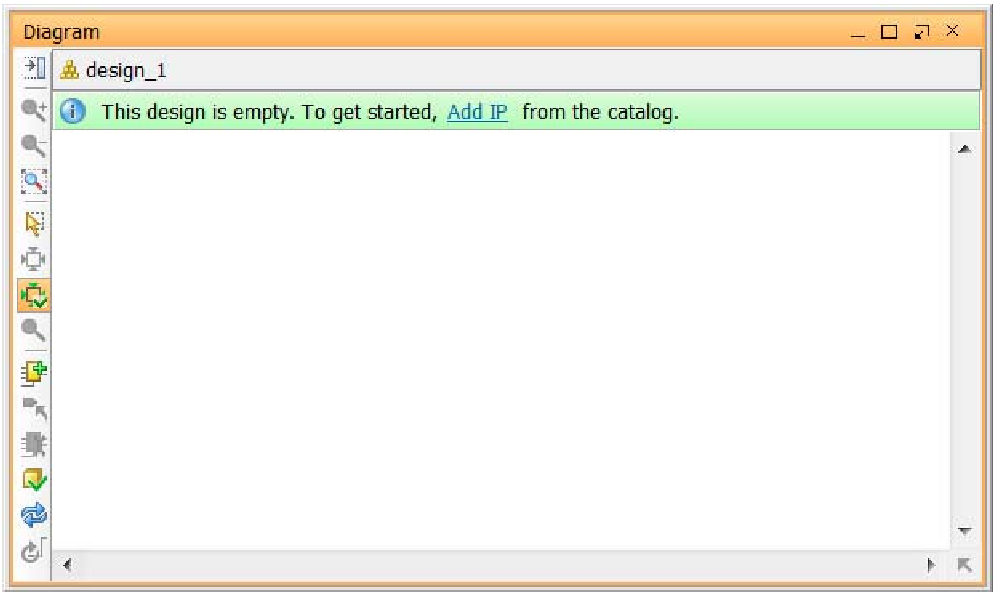
Create Block Design

Please specify name of block design

Design name: design_1

Figure 3-2: **Design Name Dialog Box**

The Block Design window opens, as shown in Figure 3-3.



Diagram

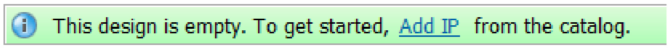
design_1

This design is empty. To get started, Add IP from the catalog.

Figure 3-3: **Block Design Window**

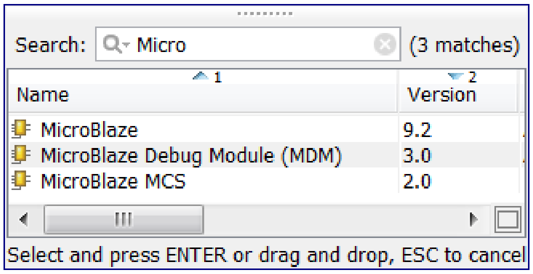
Within the empty design, there is a link to Add IP from the IP Catalog. You can also right-click in the canvas to open an option to add IP.

Click the **Add IP** link:



This design is empty. To get started, Add IP from the catalog.

A Search box opens where you can search for, and select the MicroBlaze processor, as shown in Figure 3-4.



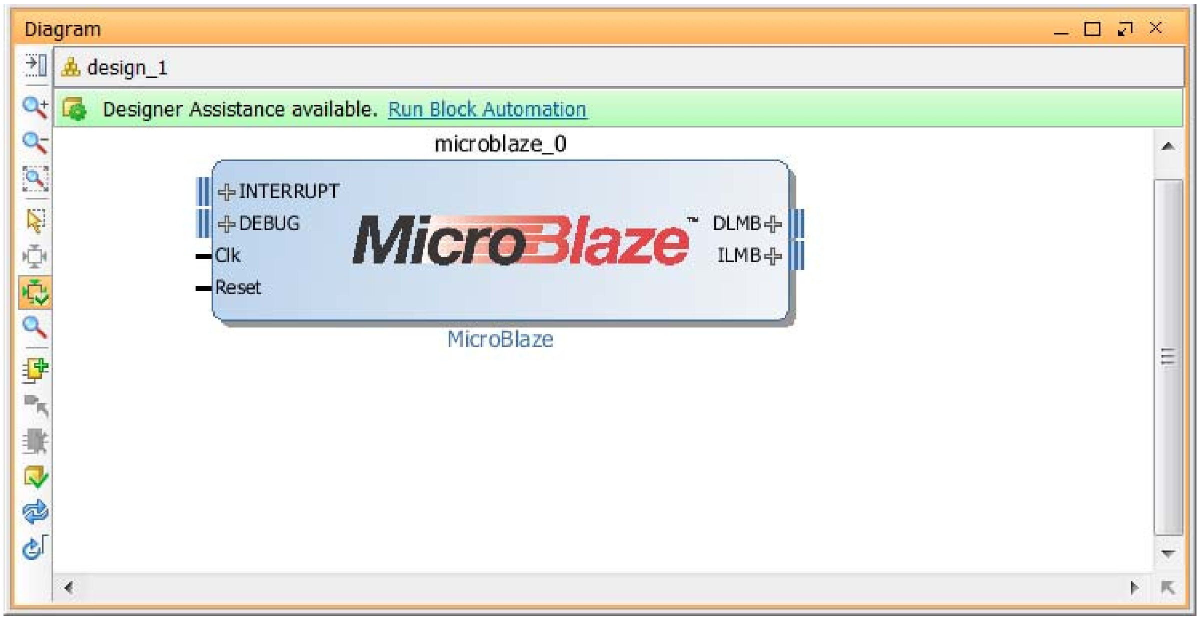
Search: Micro (3 matches)

Name	Version
MicroBlaze	9.2
MicroBlaze Debug Module (MDM)	3.0
MicroBlaze MCS	2.0

Select and press ENTER or drag and drop, ESC to cancel

Figure 3-4: **Search IP with MicroBlaze Processing System**

When you select the MicroBlaze IP, the Vivado IP integrator adds the IP to the design, and a graphical representation of the processing system displays, as shown in Figure 3-5.



Designer Assistance available. Run Block Automation

Figure 3-5: **Graphical Display of Default MicroBlaze Processing System**

Tcl Command:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:9.2 microblaze_0
```

Double-click the MicroBlaze IP in the canvas to invoke the Recustomize IP process, which displays the Re-customize IP for the MicroBlaze processor, dialog box.

MicroBlaze Configuration Window

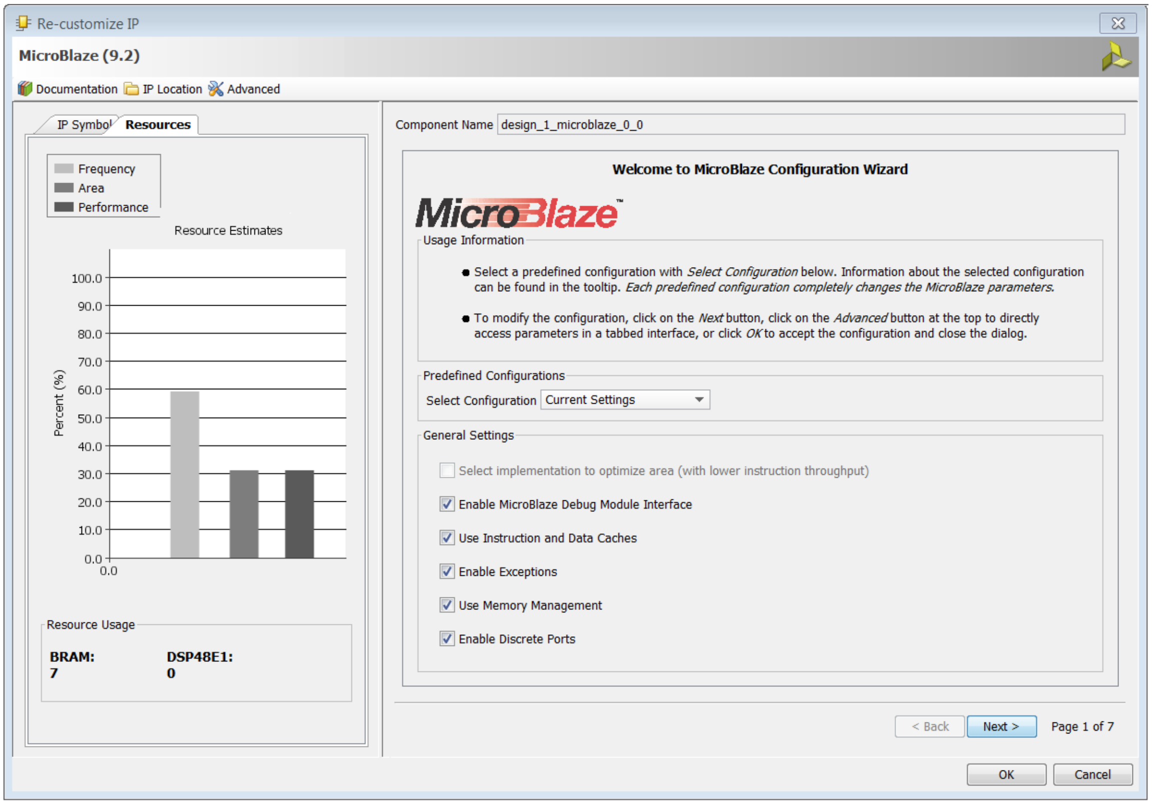
The MicroBlaze Configuration wizard provides:

- A template-based configuration dialog box for one-click configuration
- Estimates of MicroBlaze relative area, frequency, and performance, based on options set in the dialog boxes, giving immediate feedback
- Guidance through the configuration process
- Tool tips for all configuration options to understand the effect of each option
- Direct access to all options in the tabbed interface using the **Advanced** button

The MicroBlaze Configuration wizard has the following wizard pages, which enable based on the selected General Settings options:

- **Configuration Wizard**: First page that showing template selection and general settings.
- **General**: Selection of execution units, optimization that is always shown
- **Exceptions**: Exceptions to enable, which is shown if exceptions are selected on the first page
- **Debug**: Number of breakpoints and watchpoints, which is shown if debug is enabled
- **Cache**: Cache settings, which is shown if caches are selected
- **MMU**: MMU settings, which is shown if memory management is selected
- **Buses**: Bus settings. Last page, always shown

Figure 3-6 shows the Welcome page of the MicroBlaze Configuration wizard.



BRAM: 7

DSP48E1: 0

Component Name design_1_microblaze_0_0

Welcome to MicroBlaze Configuration Wizard

MicroBlaze™

Usage Information

- Select a predefined configuration with *Select Configuration* below. Information about the selected configuration can be found in the tooltip. *Each predefined configuration completely changes the MicroBlaze parameters.*
- To modify the configuration, click on the *Next* button, click on the *Advanced* button at the top to directly access parameters in a tabbed interface, or click *OK* to accept the configuration and close the dialog.

Predefined Configurations

Select Configuration: Current Settings

General Settings

- □ Select implementation to optimize area (with lower instruction throughput)
- ☑ Enable MicroBlaze Debug Module Interface
- ☑ Use Instruction and Data Caches
- ☑ Enable Exceptions
- ☑ Use Memory Management
- ☑ Enable Discrete Ports

Figure 3-6: **MicroBlaze Configuration Wizard**

The left of the dialog box shows the relative values of the frequency, area and performance for the current settings.

- **Frequency:** This value is the estimated frequency percentage relative to the maximum achievable frequency with this architecture and speed grade, which gives an indication of the relative frequency that can be achieved with the current settings.

  Note: This is an estimate based on a set of predefined benchmark systems, which can deviate up to 30% from the actual value. Do not take this estimation as a guarantee that the system can reach a corresponding frequency.

- **Area:** This value is the estimated area percentage in LUTs relative to the maximum area using this architecture, which gives an indication of the relative MicroBlaze area achievable with the current settings.

  Note: This is an estimate, which can deviate up to 5% from the actual value. Do not take this estimation as a guarantee that the implemented area matches this value.

- **Performance:** This value indicates the relative MicroBlaze performance achievable with the current settings, relative to the maximum possible performance.

  Note: This is an estimate based on a set of benchmarks, and actual performance can vary significantly depending on the user application.

- **BRAMs:** This value is the total number of block RAMs used by MicroBlaze. The instruction and data caches, and the branch target cache use block RAMS, and well as the Memory Management Unit (MMU), which uses one block RAM in virtual or protected mode.
- **DSP48** or **MULT18:** This value is the total number of DSP48 or MULT18 used by MicroBlaze. The integer multiplier, and the Floating Point Unit (FPU) use this total value to implement float multiplication.

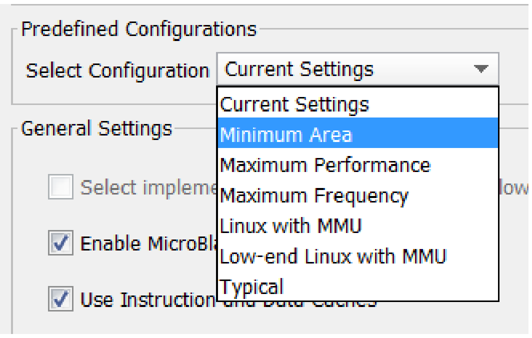
MicroBlaze Configuration Wizard Welcome Page

The simplest way to use the MicroBlaze™ Configuration wizard is to select one of the six predefined templates, each defining a complete MicroBlaze configuration. You can use a predefined template as a starting point for a specific application, using the wizard to refine the configuration, by adapting performance, frequency, or area.

When you modify an option, you received direct feedback that shows the estimated relative change in performance, frequency, and area in the information display. The options are:

- **Minimum Area:** The smallest possible MicroBlaze core. No caches or debug.
- **Maximum Performance:** Maximum possible performance. Large caches and debug, as well as all execution units.
- **Maximum Frequency:** Maximum achievable frequency. Small caches and no debug, with few execution units.
- **Linux with MMU:** Settings suitable to get high performance when running Linux with MMU. Memory Management enabled, large caches and debug, and all execution units.
- **Low-end Linux with MMU:** Settings corresponding to the MicroBlaze Embedded Reference System. Provides suitable settings for Linux development on low-end systems. Memory Management enabled, small caches and debug.
- **Typical:** Settings giving a reasonable compromise between performance, area, and frequency. Suitable for standalone programs, and low-overhead kernels. Caches and debug enabled.

Figure 3-7 shows the Predefined Configurations in the Configuration wizard.



Select Configuration: Current Settings

☐ Select impleme

☑ Enable MicroBl

☑ Use Instruction and Data Caches

Figure 3-7: **MicroBlaze Predefined Configuration Settings**

General Settings

If a pre-defined template is not used, you can select the options from the pages, which are available for fine-tuning the MicroBlaze processor, based on your design needs. As you position the mouse over these different options, a tooltip informs you what the particular option means. The following bullets detail these options.

- **Select implementation to optimize area** (with lower instruction throughput): Enables area optimized MicroBlaze. When this parameter is set, the implementation optimizes area, particularly by reducing the pipeline from five stages to three.

RECOMMENDED: *It is recommended to enable optimization on architectures with limited resources such as Artix®-7 devices. However, if performance is critical, this parameter should not be set, because some instructions require additional clock cycles to execute.*

Note: You cannot use the Memory Management Unit (MMU), Branch Target Cache, Instruction Cache Streams, Instruction Cache Victims, Data Cache Victims, and AXI Coherency Extension (ACE) with area optimization.

- **Enable MicroBlaze Debug Module Interface:** Enable debug to be able to download and debug programs using Xilinx Microprocessor Debugger.

RECOMMENDED: *Unless area resources are very critical, it is recommended that debugging be always enabled.*

- **Use Instruction and Data Caches:** You can use MicroBlaze with an optional instruction cache for improved performance when executing code that resides outside the LMB address range. The instruction cache has the following features:
  - Direct mapped (1-way associative)
  - User selectable cacheable memory address range
  - Configurable cache and tag size
  - Caching over AXI4 interface (M_AXI_IC) or CacheLink (XCL) interface
  - Option to use 4 or 8 word cacheline
  - Cache on and off controlled using a bit in the MSR
  - Optional WIC instruction to invalidate instruction cache lines
  - Optional stream buffers to improve performance by speculatively prefetching instructions
  - Optional victim cache to improve performance by saving evicted cache lines
  - Optional parity protection; invalidates cache lines if Block RAM bit error is detected
  - Optional data width selection to either use 32 bits, an entire cache line, or 512 bits

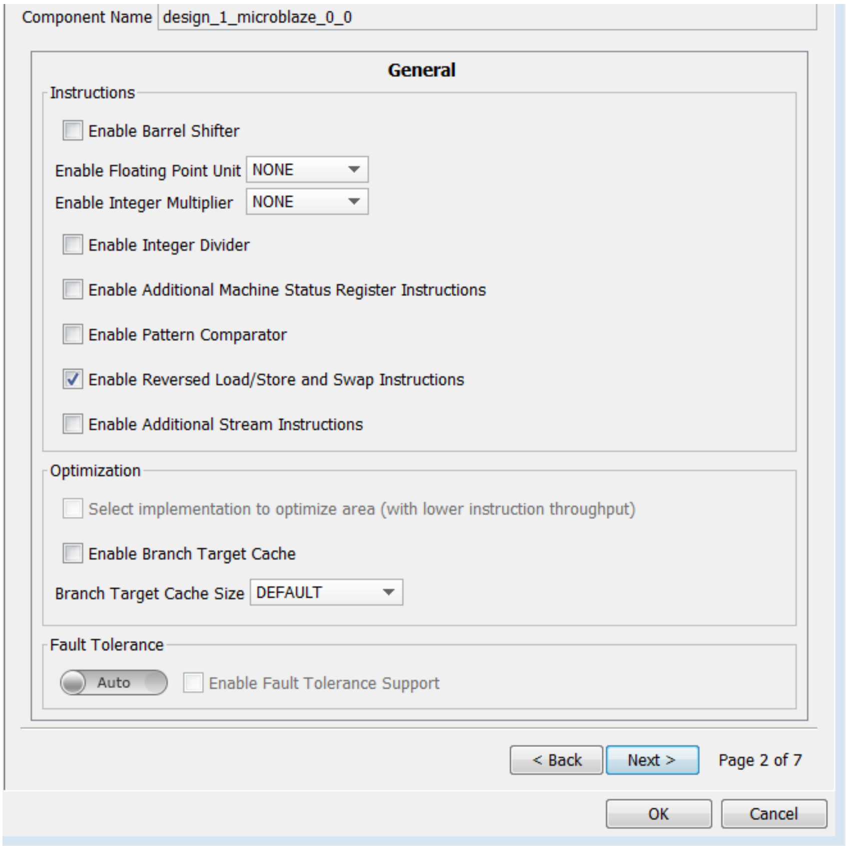
  Activating caches significantly improves performance when using external memory, even if you must select small cache sizes to reduce resource usage.

- **Enable Exceptions:** Enables exceptions when using an operating system with exception support, or when explicitly adding exception handlers in a standalone program.
- **Use Memory Management:** Enables Memory Management if planning to use an operating system - such as Linux -with support for virtual memory of memory protection.

  Note: When you enable area optimized MicroBlaze or stack protection, the Memory Management Unit is not available.
- **Enable Discrete Ports:** Enables discrete ports on the MicroBlaze instance, which is useful for:
  - Generating software breaks (Ext_BRK, Ext_NM_BRK)
  - Managing processor sleep and wakeup (Sleep, Wakeup, Dbg_Wakeup)
  - Handling debug events (Debug_Stop, MB_Halted)
  - Signaling error when using fault tolerance (MB_Error)

MicroBlaze Configuration Wizard General Page

Figure 3-8 shows the General Page of the MicroBlaze Configuration wizard.



Component Name design_1_microblaze_0_0

General

Instructions

□ Enable Barrel Shifter

Enable Floating Point Unit NONE

Enable Integer Multiplier NONE

□ Enable Integer Divider

□ Enable Additional Machine Status Register Instructions

□ Enable Pattern Comparator

☑ Enable Reversed Load/Store and Swap Instructions

□ Enable Additional Stream Instructions

Optimization

□ Select implementation to optimize area (with lower instruction throughput)

□ Enable Branch Target Cache

Branch Target Cache Size DEFAULT

Fault Tolerance

Auto □ Enable Fault Tolerance Support

Figure 3-8: **General Page of the MicroBlaze Configuration Wizard**

Instructions

- **Enable Barrel Shifter:** Enables a hardware barrel shifter in MicroBlaze. This parameter enables the instructions `bsrl`, `bsra`, `bsll`, `bsrli`, `bsrai`, and `bslli`. Enabling the barrel shifter can dramatically improve the performance of an application, but increases the size of the processor. The compiler uses the barrel shift instructions automatically if this parameter is enabled.
- **Enable Floating Point Unit:** Enables a single-precision Floating Point Unit (FPU) based on the IEEE-754 standard. Using the FPU significantly improves the single-precision, floating point performance of the application and significantly increases the size of MicroBlaze.

  Setting this parameter to BASIC enables the instructions `fadd`, `frsub`, `fmul`, `fdiv`, and `fcmp`. Setting it to **`EXTENDED`** also enables the instruction `flt`, `finit`, and `fsqrt`. The compiler automatically uses the FPU instructions corresponding to setting of this parameter.
- **Enable Integer Multiplier:** Enables a hardware integer multiplier in MicroBlaze. This parameter enables the instructions `mul` and `muli` when set to MUL32.

  When set to `MUL64`, this enables the additional instructions `mulh`, `mulhu`, and `mulhsu` for 64-bit multiplication. This parameter can be set to NONE to free up MUL or DSP48 primitives in the device for other uses. Setting this parameter to NONE has a minor effect on the area of the MicroBlaze processor. When this parameter is enabled, the compiler uses the `mul` instructions automatically.
- **Enable Integer Divider:** Enables a hardware integer divider in MicroBlaze. This parameter enables the instructions, `idiv` and `idivu`. Enabling this parameter can improve the performance of an application that performs integer division, but increases the size of the processor. When this parameter is enabled, the compiler uses the `idiv` instructions automatically.
- **Enable Additional Machine Status Register Instructions:** Enables additional machine status register (MSR) instructions for setting and clearing bits in the MSR. This parameter enables the instructions `msrset` and `msrclr`. Enabling this parameter improves the performance of changing bits in the MSR.
- **Enable Pattern Comparator:** Enables pattern compare instructions `pcmpbf`, `pcmpeq`, and `pcmpne`. The pattern compare bytes find (`pcmpbf`) instructions return the position of the first byte that matches between two words and improves the performance of string and pattern matching operations. The SDK libraries use the `pcmpbf` instructions automatically when this parameter is enabled.
  - The `pcmpeq` and `pcmpne` instructions return 1 or 0 based on the equality of the two words. These instructions improve the performance of setting flags and the compiler uses them automatically.
  - Selecting this option also enables count leading zeroes instruction, `clz`. The `clz` instruction can improve performance of priority decoding, and normalization.

- **Enable Reversed Load/Store and Swap Instructions:** Enables reversed load/store and swap instructions `lbur`, `lhur`, `lwr`, `sbr`, `shr`, `swr`, `swapb`, and `swaph`. The reversed load/store instructions read or write data with opposite endianness, and the swap instructions allow swapping bytes or half-words in registers. These instructions are mainly useful to improve performance when dealing with big-endian network access with a little-endian MicroBlaze.
- **Enable Additional Stream Instructions:** Provides additional functionality when using AXI4-Stream links, including dynamic access instruction `GETD` and `PUTD` that use registers to select the interface. The instructions are also extended with variants that provide:
  - Atomic `GET`, `GETD`, `PUT`, and `PUTD` instructions
  - Test-only `GET` and `GETD` instructions
  - `GET` and `GETD` instructions that generate a stream exception if the control bit is not set

IMPORTANT: *The stream exception must be enabled to use these instructions, and at least one stream link must be selected.*

Optimization

Select implementation to optimize area (with lower instruction throughput):This option is the same as in the General Settings options. Enable Branch Target Cache: When set, implements the branch target, which improves branch performance by predicting conditional branches and caching branch targets.

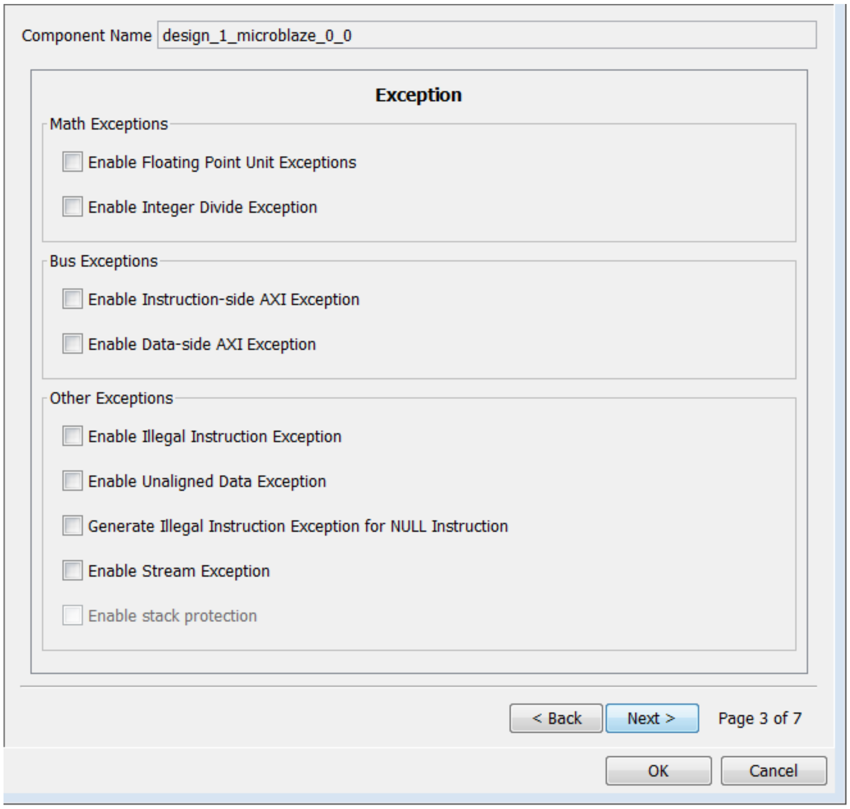
Note: To be able to use the Branch Target Cache, do not enable area optimization.

Fault Tolerence

- **Enable Fault Tolerance Support:** When enabled, MicroBlaze protects internal Block RAM with parity, and supports Error Correcting Codes (ECC) in LMB block RAM, including exception handling of ECC errors. This prevents a bit flip in block RAM from affecting the processor function.
  - If this value is auto-computed (by not overriding it), fault tolerance is automatically enabled in MicroBlaze when ECC is enabled in connected LMB BRAM controllers.
  - If fault tolerance is explicitly disabled, the IP integrator tool enables ECC automatically in connected LMB BRAM Controllers.
  - If fault tolerance is explicitly disabled, ECC in connected LMB BRAM controllers is not affected.

MicroBlaze Configuration Wizard Exception Page

Figure 3-9 shows the MicroBlaze exception options page.



Component Name: design_1_microblaze_0_0

Exception

Math Exceptions

- ☐ Enable Floating Point Unit Exceptions
- ☐ Enable Integer Divide Exception

Bus Exceptions

- ☐ Enable Instruction-side AXI Exception
- ☐ Enable Data-side AXI Exception

Other Exceptions

- ☐ Enable Illegal Instruction Exception
- ☐ Enable Unaligned Data Exception
- ☐ Generate Illegal Instruction Exception for NULL Instruction
- ☐ Enable Stream Exception
- ☐ Enable stack protection

Figure 3-9: **Exception Options in the MicroBlaze Configuration Wizard**

IMPORTANT: *You must provide your own exception handler.*

Math Exceptions

- **Enable Floating Point Unit Exceptions:** Enables exceptions generated by the Floating Point Unit (FPU). The FPU throws exceptions for all of the IEEE standard conditions: underflow, overflow, divide-by-zero, and illegal operations. In addition, the MicroBlaze FPU throws a de-normalized operand exception.
- **Enable Integer Divide Exception:** Causes an exception if the divisor (rA) provided to the idiv or idivu instruction is zero, or if an overflow occurs for idiv.

Bus Exceptions

- **Enable Instruction-side AXI Exception:** Causes an exception if there is an error on the instruction-side AXI bus.
- **Enable Data-side AXI Exception:** Causes an exception if there is an error on the data-side AXI bus.

Other Exceptions

- **Enable Illegal Instruction Exception:** Causes an exception if the major opcode is invalid.
- **Enable Unaligned Data Exception:** When enabled, the tools automatically insert software to handle unaligned accesses.
- **Generated Illegal Instruction Exception for NULL Instructions:** MicroBlaze compiler does not generate, nor do SDK libraries use the NULL instruction code (0x00000000). This code can only exist legally if it is hand-assembled. Executing a NULL instruction normally means that the processor has jumped outside the initialized instruction memory.

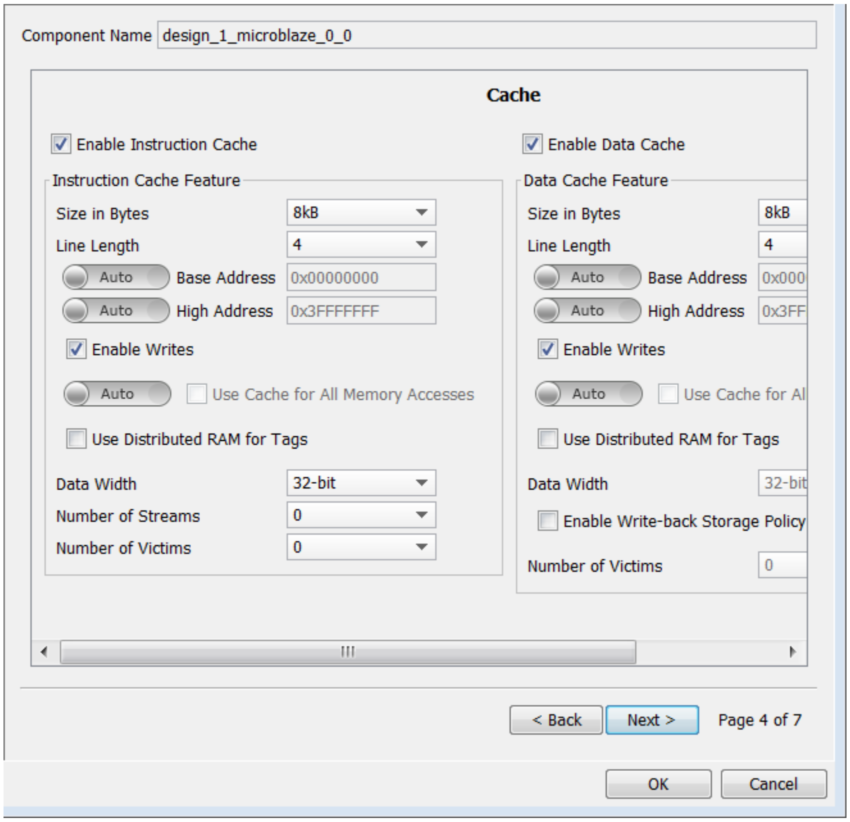
  If C_OPCODE_0x_ILLEGAL is set, MicroBlaze traps this condition; otherwise, it treats the command as a NOP. This setting is only available if you have enabled Illegal Instruction Exception.
- **Enable Stream Exception:** Enables stream exception handling for Advanced eXtensible Interface (AXI) read accesses.

IMPORTANT: *You must enable additional stream instructions to use stream exception handling.*

- **Enable Stack Protection:** Ensures that memory accesses using the stack pointer (R1) to ensure they are within the limits set by the Stack Low Register (SLR) and Stack High Register (SHR). If the check fails with exceptions enabled, a Stack Protection Violation exception occurs. The Xilinx Microprocessor Debugger (XMD) also reports if the check fails.

MicroBlaze Configuration Wizard Cache Page

Figure 3-10 shows the Cache options page for the MicroBlaze Configuration.



Component Name design_1_microblaze_0_0

Cache

☑ Enable Instruction Cache

Instruction Cache Feature

Size in Bytes 8kB

Line Length 4

Auto Base Address 0x00000000

Auto High Address 0x3FFFFFFF

☑ Enable Writes

Auto □ Use Cache for All Memory Accesses

□ Use Distributed RAM for Tags

Data Width 32-bit

Number of Streams 0

Number of Victims 0

☑ Enable Data Cache

Data Cache Feature

Size in Bytes 8kB

Line Length 4

Auto Base Address 0x000

Auto High Address 0x3FF

☑ Enable Writes

Auto □ Use Cache for Al

□ Use Distributed RAM for Tags

Data Width 32-bit

□ Enable Write-back Storage Policy

Number of Victims 0

< Back Next > Page 4 of 7

OK Cancel

Figure 3-10: **Cache Options Page of the MicroBlaze Configuration Wizard**

- **Enable Instruction Cache:** Uses this cache only when it is also enabled in software by setting the instruction cache enable (ICE) bit in the machine status register (MSR).
- Instruction Cache Features:
  - **Size in Bytes:** Specifies the size of the instruction cache if C_USE_ICACHE is enabled. Not all architectures permit all sizes.
  - **Line Length:** Select between 4 or 8 word cache line length for cache miss-transfers from external instruction memory.
  - **Base Address:** Specifies the base address of the instruction cache. This parameter is used only if C_USE_ICACHE is enabled.
  - **High Address:** Specifies the high address of the instruction cache. This parameter is used only if C_USE_ICACHE is enabled.
  - **Enable Writes:** When enabled, one can invalidate instruction cache lines with the wic instruction. This parameter is used only if C_USE_ICACHE is enabled.

- **Use Cache for All Memory Accesses:** When enabled, uses the dedicated cache interface on MicroBlaze is for all accesses within the cacheable range to external instruction memory, even when the instruction cache is disabled.

  Otherwise, the instruction cache uses the peripheral AXI for these accesses when the instruction cache is disabled. When enabled, an external memory controller must provide only a cache interface MicroBlaze instruction memory. Enable this parameter when using AXI Coherency Extension (ACE).

- **Use Distributed RAM for Tags:** Uses the instruction cache tags to hold the address and a valid bit for each cacheline. When enabled, the instruction cache tags are stored in Distributed RAM instead of Block RAM. This saves Block RAM, and can increase the maximum frequency.

- **Data Width:** Specifies the instruction cache bus width when using AXI Interconnect. The width can be set to:

- **32-bit:** Bursts are used to transfer cache lines for 32-bit words depending on the cache line length,

- **Full Cacheline:** A single transfer is performed for each cache line, with data width 128 or 256 bits depending on cache line length

- **512-bit:** Performs a single transfer, but uses only 128 or 256 bits, depending on cacheline length.

  The two wide settings require that the cache size is at least 8 KB or 16KB depending upon cache line length. To reduce the AXI interconnect size, this setting must match the interconnect data width. In most cases, you can obtain the best performance with the wide settings.

  Note: This setting is not available with area optimization, AXI Coherency Extension (ACE), or when you enable fault tolerance.

- **Number of Streams:** Specifies the number of stream buffers used by the instruction cache. A stream buffer is used to speculatively pre-fetch instructions, before the processor requests them. This often improves performance, because the processor spends less time waiting for instruction to be fetched from memory.

  Note: To be able to use instruction cache streams, do not enable area optimization or AXI Coherency Extension (ACE).

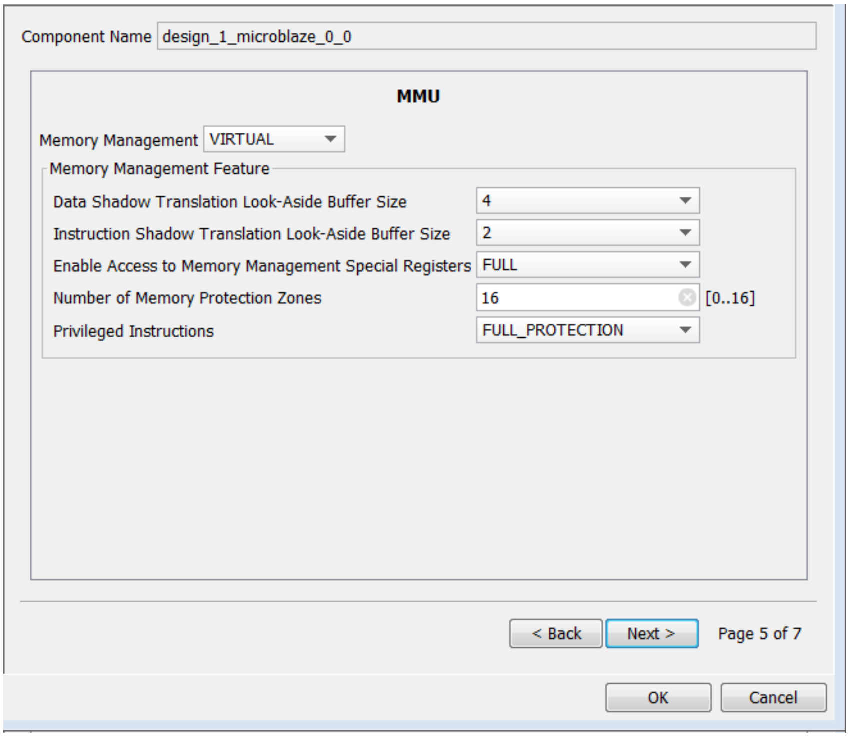
- **Number of Victims:** Specifies the number of instruction cache victims to save. A victim is a cacheline that is evicted from the cache. If no victims are saved, all evicted lines must be read from memory again, when they are needed. By saving the most recent lines, they can be fetched much faster, thus improving performance.

RECOMMENDED: *It is possible to save 2, 4, or 8 cachelines. The more cachelines that are saved, the better performance becomes. The recommended value is 8 lines.*

Note: To be able to use instruction cache victims, do not enable area optimization or AXI Coherency Extension (ACE).

MicroBlaze Configuration Wizard MMU Page

Figure 3-11 shows the MMU page of the MicroBlaze Configuration.



Component Name design_1_microblaze_0_0

MMU

Memory Management VIRTUAL

Memory Management Feature

Data Shadow Translation Look-Aside Buffer Size 4

Instruction Shadow Translation Look-Aside Buffer Size 2

Enable Access to Memory Management Special Registers FULL

Number of Memory Protection Zones 16 [0..16]

Privileged Instructions FULL_PROTECTION

Page 5 of 7

Figure 3-11: **MicroBlaze Configuration Wizard MMU Page**

Memory Management

Memory management specifies the Memory Management Unit (MMU) implementation.

To disable the MMU, set this parameter to None (0), the default.

- To enable only the User Mode and Privileged Mode instructions, set this parameter to Usermode (1). To enable Memory Protection, set the parameter to Protection (2).
- To enable full MMU functionality, including virtual memory address translation, set this parameter to Virtual (3).

When Usermode is set, it enables the Privileged Instruction exception. When Protection or Virtual is set, it enables the Privileged Instruction exception and the four MMU exceptions (Data Storage, Instruction Storage, Data TLB Miss, and Instruction TLB Miss).

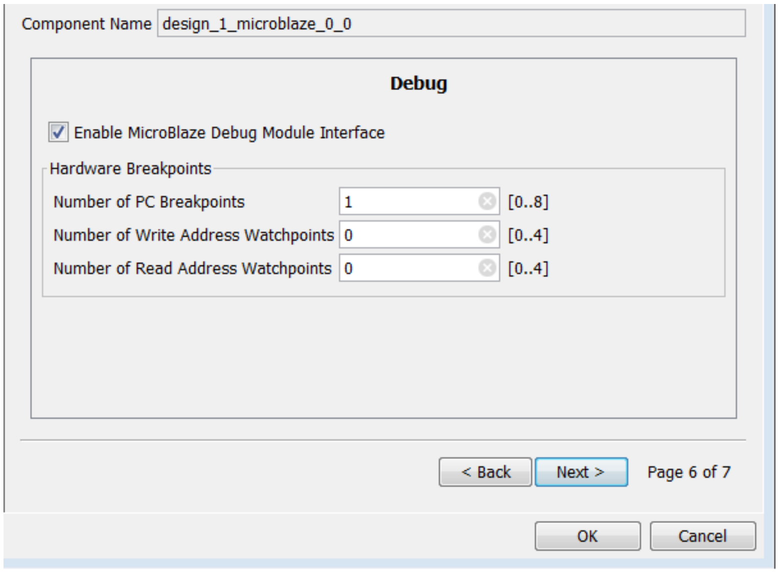
Memory Management Features:

- **Data Shadow Translation Look-Aside Buffer Size:** Defines the size of the instruction shadow Translation Look-Aside Buffer (TLB). This TLB caches data address translation information, to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 4.
- **Instruction Shadow Translation Look-Aside Buffer Size:** Defines the size of the instruction shadow Translation Look-Aside Buffer (TLB). This TLB caches instruction address translation information to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 2.
- **Enable Access to Memory Management Special Registers:** Enables access to the Memory Management Special Register using the MFS and MTS instructions:
  - Minimal (0) only allows writing TLBLO, TLBHI, and TLBX.
  - Read (1) adds reading to TLBLO, TLBHI, TLBX, PID, and ZPR.
  - Write (2) allows writing all registers, and reading TLBX.
  - Full (3) adds reading of TLBLO, TLBHI, TLBX, PID, and ZPR.

  In many cases, it is not necessary for the software to have full read access. For example, this is the case for Linux Memory Management code. It is then safe to set access to Write, to save area. When using static memory protection, access can be set to Minimal, because the software then has no need to use TLBSX, PID, and ZPR.
- **Number of Memory Protection Zones:** Specifies the number of memory protection zones to implement. In many cases memory management software does not use all available zones. For example, the Linux Memory Management code only uses two zones. In this case, it is safe to reduce the number of implemented zones, to save area.
- **Privileged Instructions:** Specifies which instructions to allow in User Mode.
  - The Full Protection (0) setting ensures full protection between processes.
  - The Allow Stream Instructions (1) setting makes it possible to use AXI4-Stream instructions in User Mode.

CAUTION! *It is strongly discouraged to change this setting from Full Protection, unless it is necessary for performance reasons.*

MicroBlaze Configuration Wizard Debug Page



Component Name: design_1_microblaze_0_0

Debug

☑ Enable MicroBlaze Debug Module Interface

Hardware Breakpoints

Number of PC Breakpoints	1	[0..8]
Number of Write Address Watchpoints	0	[0..4]
Number of Read Address Watchpoints	0	[0..4]

Figure 3-12: **MicroBlaze Configuration Wizard Debug Page**

Debug Options

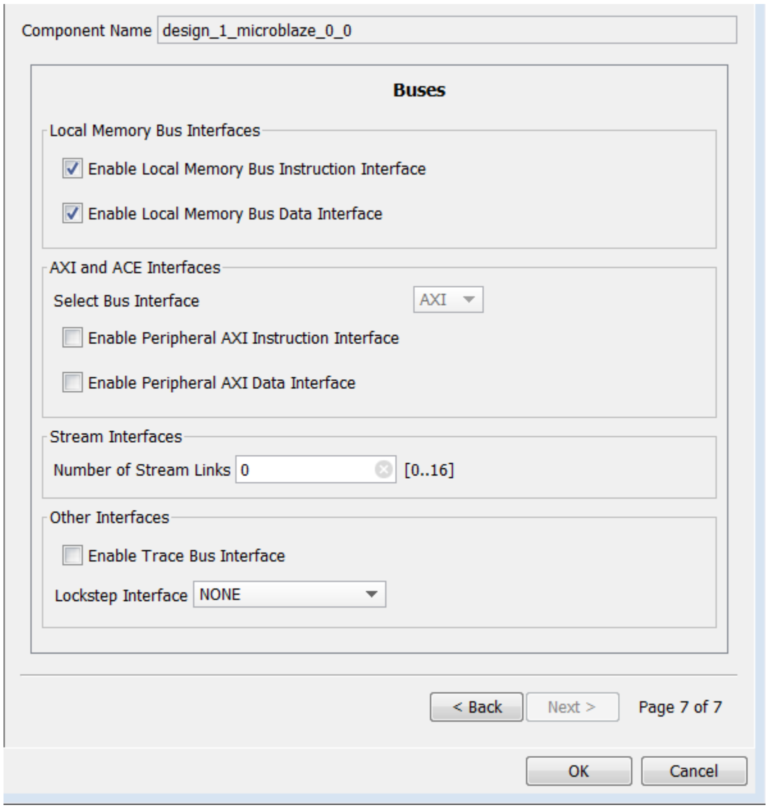
Enable MicroBlaze Debug Module Interface: Enables the MicroBlaze Debug Module (MDM) interface to MicroBlaze for debugging. With this option, you can use Xilinx Microprocessor Debugger (XMD) to debug the processor over the Joint Test Action Group (JTAG) boundary-scan interface. You can disable this option after you finish debugging to reduce the size of MicroBlaze.

Hardware Breakpoints:

- **Number of PC Breakpoints:** Specifies the number of program counter (PC) hardware breakpoints for debugging. This parameter controls the number of hardware breakpoints Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if C_DEBUG_ENABLED is on. The MicroBlaze processor takes a noticeable frequency hit the larger this parameter is set.
- **Number of Write Address Watchpoints:** Specifies the number of write address breakpoints for debugging. This parameter controls the number of write watchpoints Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if C_DEBUG_ENABLED is on. MicroBlaze take a noticeable frequency hit, the larger this parameter is set.
- **Number of Read Address Watchpoints:** Specifies the number of read address breakpoints for debugging. This parameter controls the number of read watch points Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if C_DEBUG_ENBLED is on. The MicroBlaze processor takes a noticeable frequency hit the larger this parameter is set.

RECOMMENDED: *It is recommended that these two options be set to 0 if you are not using watch points for debugging.*

MicroBlaze Configuration Wizard Buses Page



Component Name: design_1_microblaze_0_0

Buses

Local Memory Bus Interfaces

☑ Enable Local Memory Bus Instruction Interface

☑ Enable Local Memory Bus Data Interface

AXI and ACE Interfaces

Select Bus Interface: AXI

□ Enable Peripheral AXI Instruction Interface

□ Enable Peripheral AXI Data Interface

Stream Interfaces

Number of Stream Links: 0 [0..16]

Other Interfaces

□ Enable Trace Bus Interface

Lockstep Interface: NONE

Figure 3-13: **MicroBlaze Configuration Wizard Buses Page**

Local Memory Bus Interfaces:

- **Enable Local Memory Bus Instruction Interface:** Enables LMB instruction interface. When this instruction is set, the Local Memory Bus (LMB) instruction interface is available. A typical MicroBlaze system uses this interface to provide fast local memory for instructions. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common Block RAM.
- **Enable Local Memory Bus Data Interface:** Enables LMB data interface. When this parameter is set, the Local Memory Bus (LMB) data interface is available. A typical MicroBlaze system uses this interface to provide fast local memory for data and vectors. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common Block RAM.

AXI and ACE Interfaces:

- **Select Bus Interface:** When this parameter is set to AXI, then AXI is selected for both peripheral and cache access. When this parameter is set to ACE, then AXI is selected for peripheral access and AXI Coherency Extension (ACE) is selected for cache access, providing cache coherency support.

  Note: To be able to use ACE, area optimization, write-back data cache, instruction cache streams or victims, and cache data widths other than 32-bit must not be set. You must set Use Cache for All Memory Accesses for both caches.

- **Enable Peripheral AXI Interface Instruction Interface:** When this parameter is set, the peripheral AXI4-Lite instruction interface is available. In many cases, this interface is not needed, in particular if the Instruction Cache is enabled and C_ICACHE_ALWAYS_USED is set.
- **Enable Peripheral AXI Data Interface:** When this parameter is set, the peripheral AXI data interface is available. This interface usually connects to peripheral I/O using AXI4-Lite, but it can be connected to memory also. If you enable exclusive access, the AXI4 protocol is used.

Stream Interfaces:

- **Number of Stream Links:** Specifies the number of pairs of AXI4-Stream link interfaces. Each pair contains a master and a slave interface. The interface provides a unidirectional, point-to-point communication channel between MicroBlaze and a hardware accelerator or coprocessor. This is a low-latency interface, which provides access between the MicroBlaze register file and the FPGA fabric.

Other Interfaces:

- **Enable Trace Bus Interface:** When this parameter is set, the Trace bus interface is available. This interface is useful for debugging, execution statistics and performance analysis. In particular, connecting interface to a ChipScope™ Logic Analyzer (ILA) allows tracing program execution with clock cycle accuracy.
- **Lockstep Interface:** When you enable lockstep support, two MicroBlaze cores run the same program in lockstep, and you can compare their outputs to detect errors.
  - When set to **`NONE`** , no lockstep interfaces are enabled.
  - When set to **`LOCKSTEP_MASTER`**, it enables the `Lockstep_Master_Out` and `Lockstep_Out` output ports.
  - When set to **`LOCKSTEP_SLAVE`**, it enables the `Lockstep_Slave_in` input port and `Lockstep_Out` output ports, and the `C_LOCSTEP_SLAVE` parameter is set to 1.

Custom Logic

The Vivado IP packager lets you and third party IP developers use the Vivado IDE to prepare an Intellectual Property (IP) design for use in the Vivado IP catalog. The IP user can then instantiate this third party IP into a design in the Vivado Design Suite.

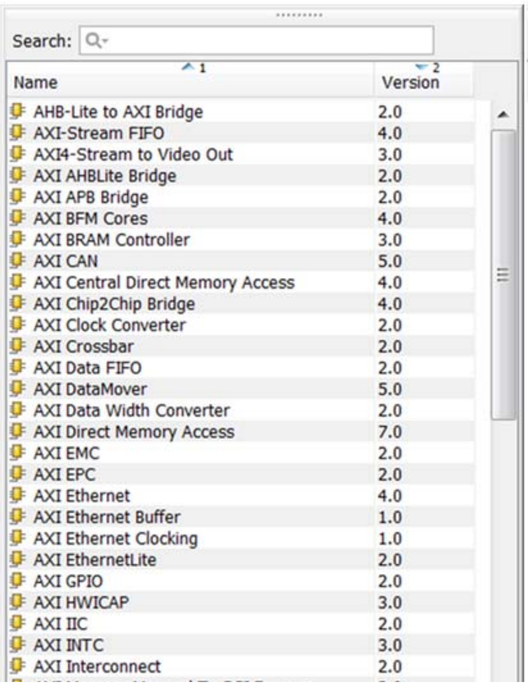
When IP developers use the Vivado Design Suite IP packaging flow, the IP user has a consistent experience whether using Xilinx IP, third party IP, or customer-developed IP within the Vivado Design Suite.

IP developers can use the IP packager feature to package IP files and associated data into a ZIP file. The IP user receives this generated ZIP file, installs the IP into the Vivado Design Suite IP Catalog. The IP user then customizes the IP through parameter selections and generates an instance of the IP.

RECOMMENDED: *To verify the proper packaging of the IP before handing it off to the IP user, Xilinx recommends that the IP developer run each IP module completely through the IP user flow to verifiy that the IP is ready for use.*

Embedded IP Catalog

The Vivado IDE IP Catalog is a unified repository that lets you search, review detailed information, and view associated documentation for the IP. After you add the third party or customer IP to the Vivado Design Suite IP catalog, you can access the IP through the Vivado Design Suite flows. Figure 3-14 shows a portion of the Vivado IDE IP Catalog.



Name	Version
AHB-Lite to AXI Bridge	2.0
AXI-Stream FIFO	4.0
AXI4-Stream to Video Out	3.0
AXI AHBLite Bridge	2.0
AXI APB Bridge	2.0
AXI BFM Cores	4.0
AXI BRAM Controller	3.0
AXI CAN	5.0
AXI Central Direct Memory Access	4.0
AXI Chip2Chip Bridge	4.0
AXI Clock Converter	2.0
AXI Crossbar	2.0
AXI Data FIFO	2.0
AXI DataMover	5.0
AXI Data Width Converter	2.0
AXI Direct Memory Access	7.0
AXI EMC	2.0
AXI EPC	2.0
AXI Ethernet	4.0
AXI Ethernet Buffer	1.0
AXI Ethernet Clocking	1.0
AXI EthernetLite	2.0
AXI GPIO	2.0
AXI HWICAP	3.0
AXI IIC	2.0
AXI INTC	3.0
AXI Interconnect	2.0

Figure 3-14: **IP Integrator IP Catalog**

Completing Connections

After you have configured the MicroBlaze processor, you can start to instantiate other IP that constitutes your design.

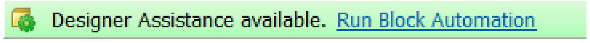
In the IP Integrator canvas, right-click and select **Add IP**.

You can use two built-in features of the IP integrator to complete the rest of the IP subsystem design: the Block Automation and Connection Automation features assist you with putting together a basic microprocessor system in the IP integrator tool and/or connecting ports to external I/O ports.

Block Automation

The Block Automation feature is available when a microprocessor such as the ZYNQ7 Processing System (PS) or the MicroBlaze Processor is instantiated in the block design of the IP integrator tool.

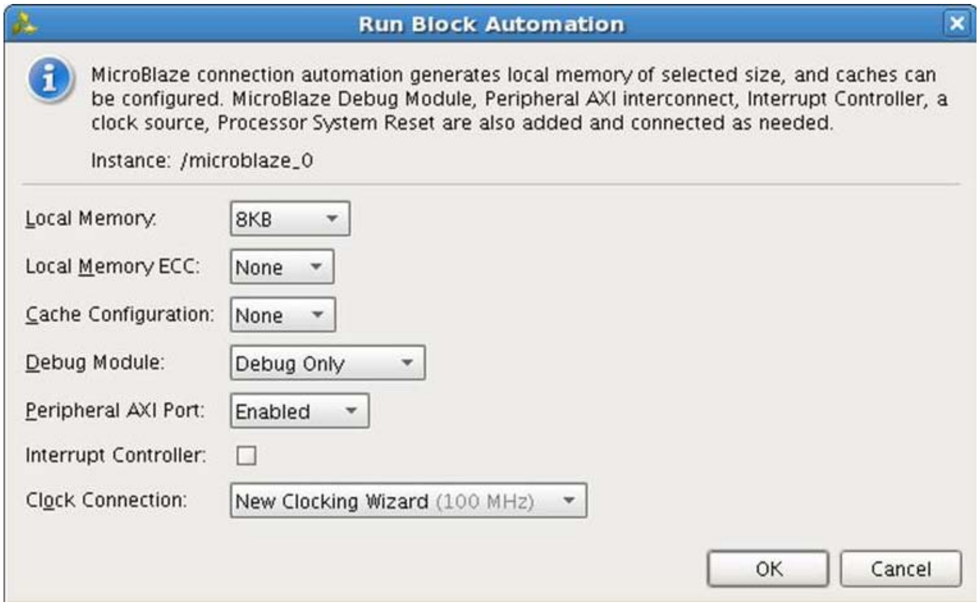
1. Click **Run Block Automation** to get assistance with putting together a simple MicroBlaze System.



Designer Assistance available. Run Block Automation

Figure 3-15: **Designer Assistance: Run Block Automation**

The Run Block Automation dialog box lets you provide input about basic features that the microprocessor system requires.



Run Block Automation

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Local Memory: 8KB

Local Memory ECC: None

Cache Configuration: None

Debug Module: Debug Only

Peripheral AXI Port: Enabled

Interrupt Controller: □

Clock Connection: New Clocking Wizard (100 MHz)

Figure 3-16: **Run Block Automation for MicroBlaze Processor Dialog Box**

2. Select the required options and click **OK**.

   Run Block Automation creates the following MicroBlaze system.

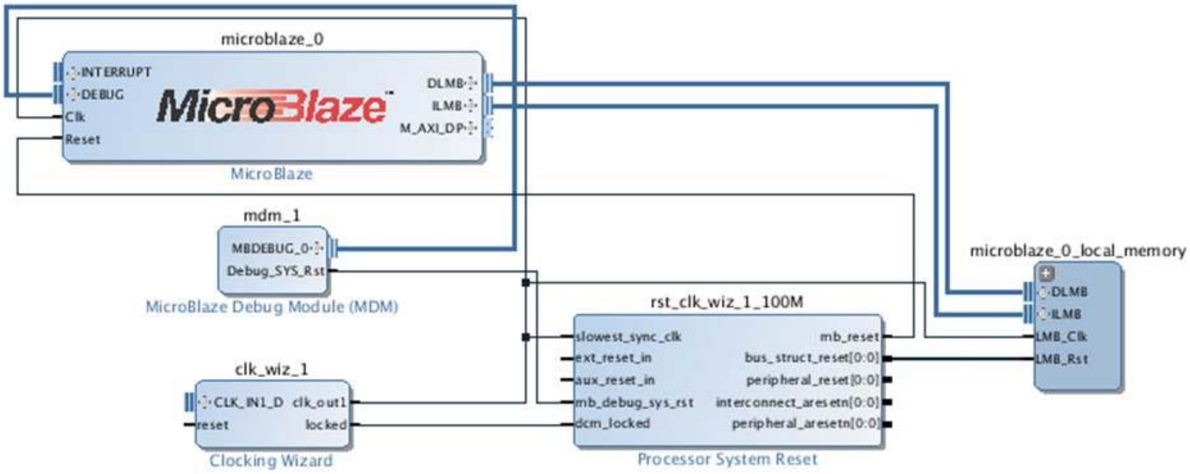


Figure 3-17: **IP Integrator Window After Running Block Automation**

Using Connection Automation

When the IP integrator tool determines that a potential connection exists among the instantiated IP in the canvas, it opens the Connection Automation feature.

In Figure 3-18, two IP, the GPIO and the Uartlite, are instantiated along with the MicroBlaze subsystem.

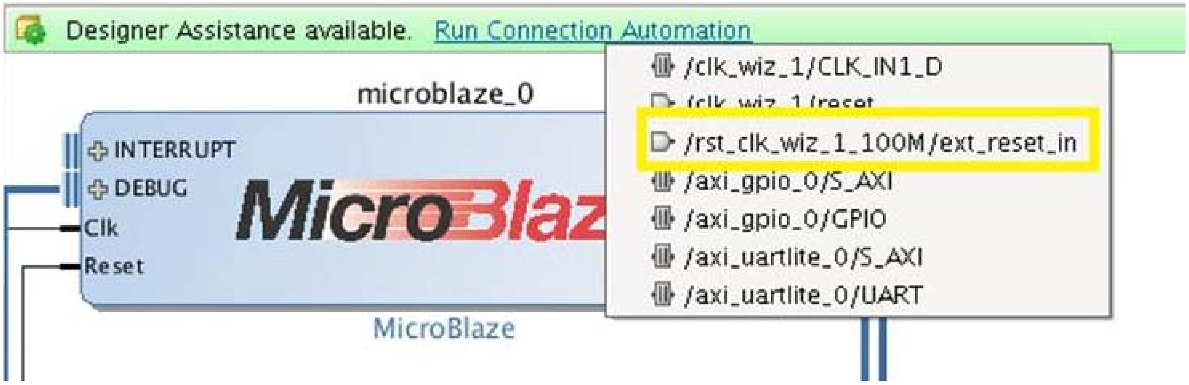

Designer Assistance available. Run Connection Automation

Figure 3-18: **Connection Automation Feature of IP Integrator**

The IP integrator determines that there is a potential connection for the following objects:

- The Proc Sys Rst IP ext_reset_in pin must connect to a reset source, which can be either an internal reset source or an external input port.
- The Clocking Wizard CLK_IN_1_D pin must connect to either an internal clock source or an external input port.
- The AXI GPIO s_axi interface must connect to a master AXI interface.
- The AXI GPIO core gpio interface must connect to external I/Os.
- The Uartlite IP s_axi interface must connect to a master AXI interface.
- The Uartlite IP uart interface must connect to external I/Os.

When you run connection automation on each of those available options, the block design looks like Figure 3-19.

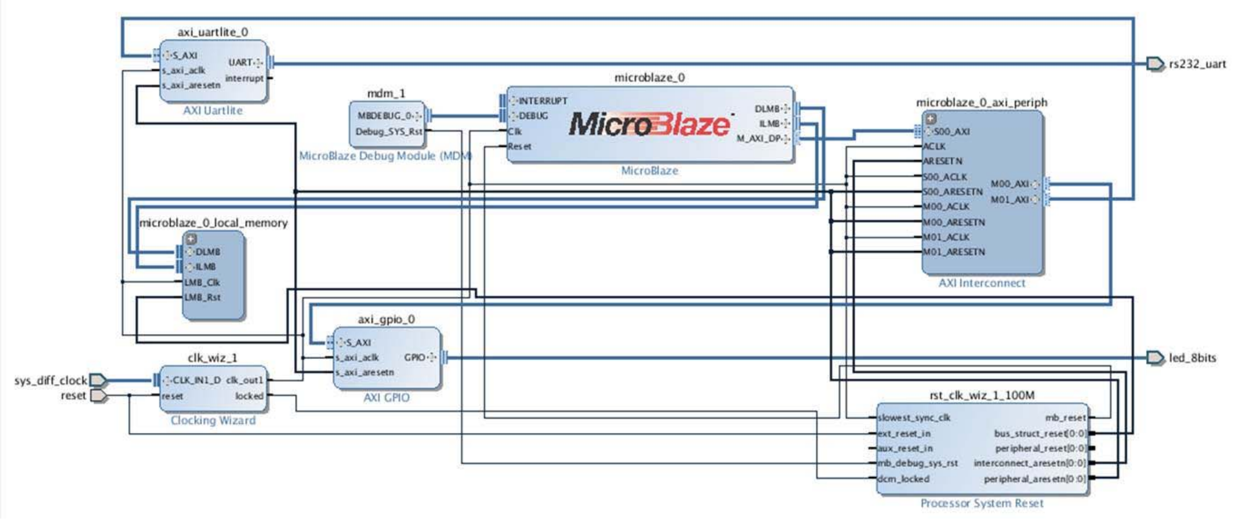


Figure 3-19: **Running Connection Automation on MicroBlaze Design**

Using Board Automation

The Vivado IDE IP Integrator tool also provides a Board Automation feature when using a Xilinx Target Reference Platform, such as the KC705.

This feature provides connectivity of the ports of IP to the FPGA pins on the target board. The IP configures accordingly, and, based upon your selections, connects the I/O ports. The Board Automation feature also generates the physical constrains for those IP that require physical constraints. These features are available to you via the Run Connection Automation feature of the IP integrator tool.

As an example, for the GPIO core, when the target board is KC705, the GPIO interface can connect to one of the following:

- DIP Switches
- LEDs
- Push Buttons

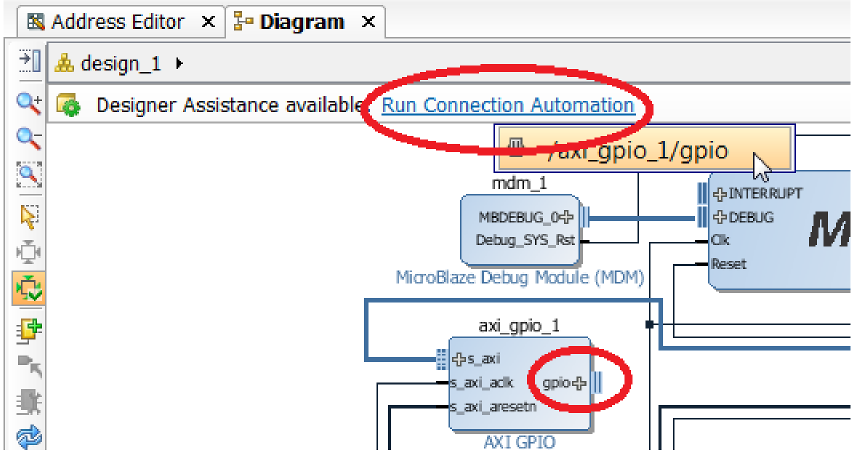
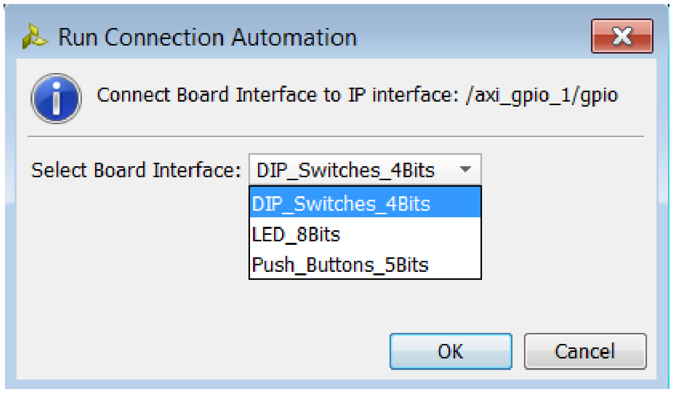


Figure 3-20: **Board Automation on GPIO Port**

Running connection automation for the gpio interface gives you the following options in the **Select Board Interface** drop-down menu.


Run Connection Automation

Connect Board Interface to IP interface: /axi_gpio_1/gpio

Select Board Interface: DIP_Switches_4Bits

Figure 3-21: **Run Connection Automation Options for GPIO**

Depending on the selected option, the GPIO IP is not only configured accordingly (as inputs or outputs), but the appropriate sets of physical constraints are also generated.

Manual Connections in an IP Integrator Design

See Manual Connections in a Design, page 36.

Manually Creating and Connecting to I/O Ports

See Manually Creating and Connecting to I/O Ports, page 36.

Memory Mapping in Address Editor

See Memory Mapping in Address Editor, page 37.

Running Design Rule Checks

See Running Design Rule Checks, page 37.

Integrating a Block Design in the Top-Level Design

See Integrating a Block Design in the Top-Level Design, page 37.

MicroBlaze Processor Constraints

The IP integrator generates constraints for IP generated within the tool during output products generation; however, you must generate constraints for any custom IP or higher-level code.

A constraint set is a set of XDC files that contain design constraints, which you can apply to your design. There are two types of design constraints:

- Physical constraints define pin placement, and absolute, or relative placement of cells such as: BRAMs, LUTs, Flip Flops, and device configuration settings.
- Timing constraints, written in industry standard SDC, define the frequency requirements for the design. Without timing constraints, the Vivado Design Suite optimizes the design solely for wire length and routing congestion.

  Note: Without timing constraints, Vivado implementation makes no effort to assess or improve the performance of the design.

IMPORTANT: *The Vivado Design Suite does not support UCF format. For information on migrating UCF constraints to XDC commands refer to the Vivado Design Suite User Guide [Ref 7] for more information.*

You have a number of options on how to use constraint sets. You can have:

- Multiple constraints files within a constraint set.
- Constraint sets with separate physical and timing constraint files.
- A master constraints file, and direct design changes to a new constraints file.
- Multiple constraint sets for a project, and make different constraint sets active for different implementation runs to test different approaches.
- Separate constraint sets for synthesis and for implementation.
- Different constraint files to apply during synthesis, simulation, and implementation to help meet your design objectives.

Separating constraints by function into different constraint files can make your overall constraint strategy more clear, and facilitate being able to target timing and implementation changes.

Organizing design constraints into multiple constraint sets can help you do the following:

- Target different Xilinx FPGAs for the same project. Different physical and timing constraints could be necessary for different target parts.
- Perform "what-if" design exploration. Using constraint sets to explore different scenarios for floorplanning and over-constraining the design.
- Manage constraint changes. Override master constraints with local changes in a separate constraint file.

TIP: *A good way to validate the timing constraints is to run the report_timing_summary command on the synthesized design. Problematic constraints must be addressed before implementation.*

For more information on defining and working with constraints that affect placement and routing, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 8].

Taking the Design through Synthesis, Implementation and Bitstream Generation

After you complete the design and constrain it appropriately, you can run synthesis and implementation, and then you can generate a bitstream.

Exporting Hardware to the Software Development Kit (SDK)

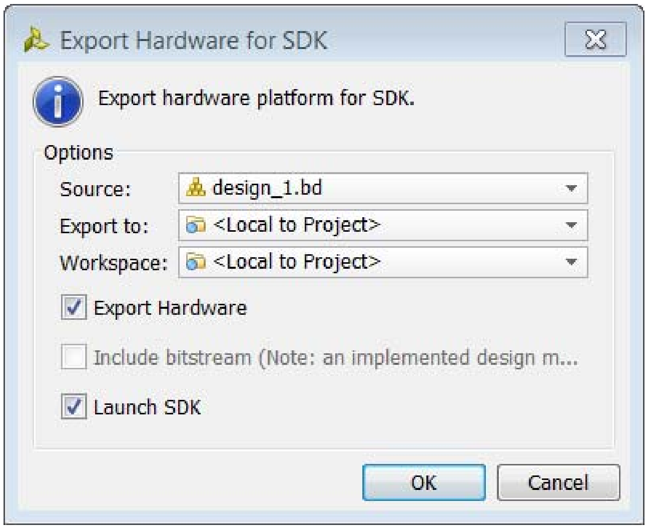
See Using the Software Development Kit (SDK), page 39 for more information.

In general, after you generate the bitstream for your design, you are ready to export your hardware definition to SDK.

Select **File > Export > Export Hardware for SDK**.

This launches the Export Hardware for SDK dialog box, where you can choose the available export options.

You can export the hardware definition and the bitstream, and launch SDK through the Export Hardware for SDK dialog box.



Export Hardware for SDK

Export hardware platform for SDK.

Options

Source: design_1.bd

Export to: <Local to Project>

Workspace: <Local to Project>

☑ Export Hardware

□ Include bitstream (Note: an implemented design m...

☑ Launch SDK

Figure 3-22: **Export Hardware for SDK Dialog Box**

After you export the hardware definition to SDK, and launch SDK, you can start writing your software application. Also, you can perform more debug and software from SDK.

Alternatively, you can import the software ELF file back into a Vivado IDE project, and integrate that file with an FPGA bitstream for further download and testing.

Chapter 4

Designing with the MIG Core

Overview

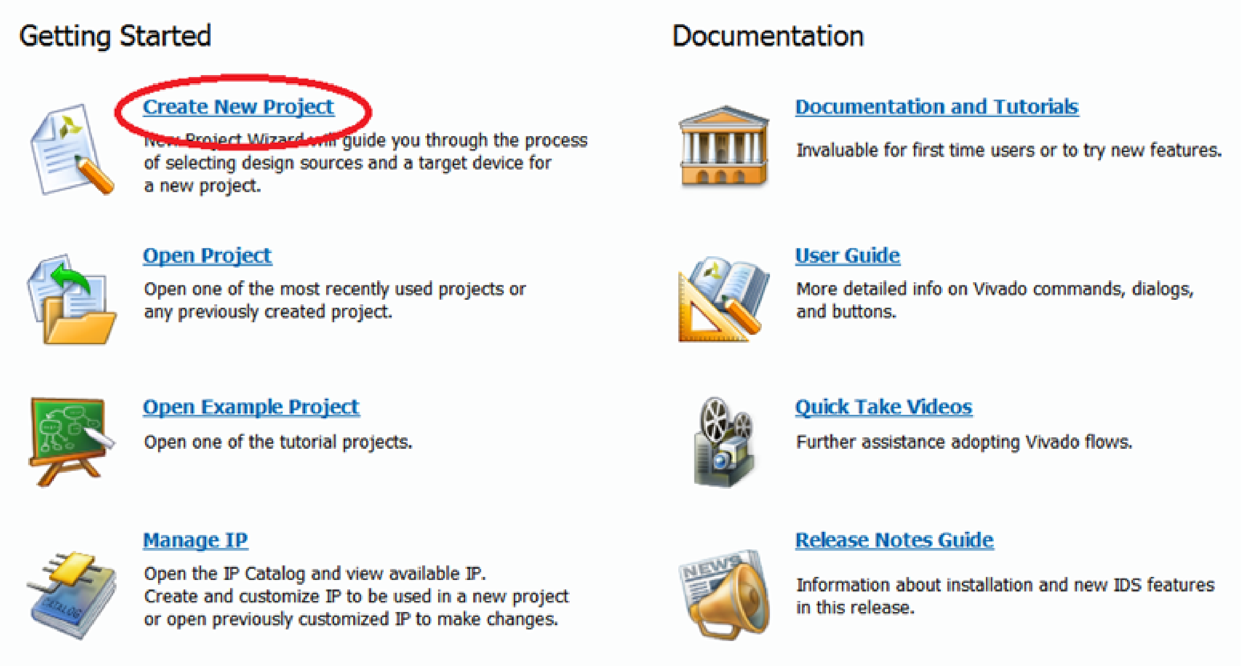
The Xilinx® 7 series FPGAs memory interface generator (MIG) core is a combined pre-engineered controller and physical layer (PHY) for interfacing 7 series FPGA user designs and AMBA® advanced extensible interface (AXI4) slave interfaces to DDR3 and DDR2 SDRAM devices.

This chapter provides information about using, customizing, and simulating a LogiCORE™ IP DDR3 or DDR2 SDRAM interface core for 7 series FPGAs in the IP integrator tool. In the Embedded Development Kit (EDK), this core is provided through the Xilinx Platform Studio (XPS) as the axi_7series_ddrx IP with a static AXI4 to DDR3 or DDR2 SDRAM architecture. The chapter describes the core architecture and provides details on customizing and interfacing to the core.

Although the information in this chapter is tailored for the KC705, Kintex-7 board, these guidelines can also be applied to custom user hardware.

Project Creation

Although you can create entire designs using IP integrator, a typical design consists of HDL, IP, and IP integrator block designs. In the Vivado® tools, you can create a new design using Create New Project wizard, as shown in Figure 4-1.



Getting Started

Create New Project

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Open Project

Open one of the most recently used projects or any previously created project.

Open Example Project

Open one of the tutorial projects.

Manage IP

Open the IP Catalog and view available IP. Create and customize IP to be used in a new project or open previously customized IP to make changes.

Documentation

Documentation and Tutorials

Invaluable for first time users or to try new features.

User Guide

More detailed info on Vivado commands, dialogs, and buttons.

Quick Take Videos

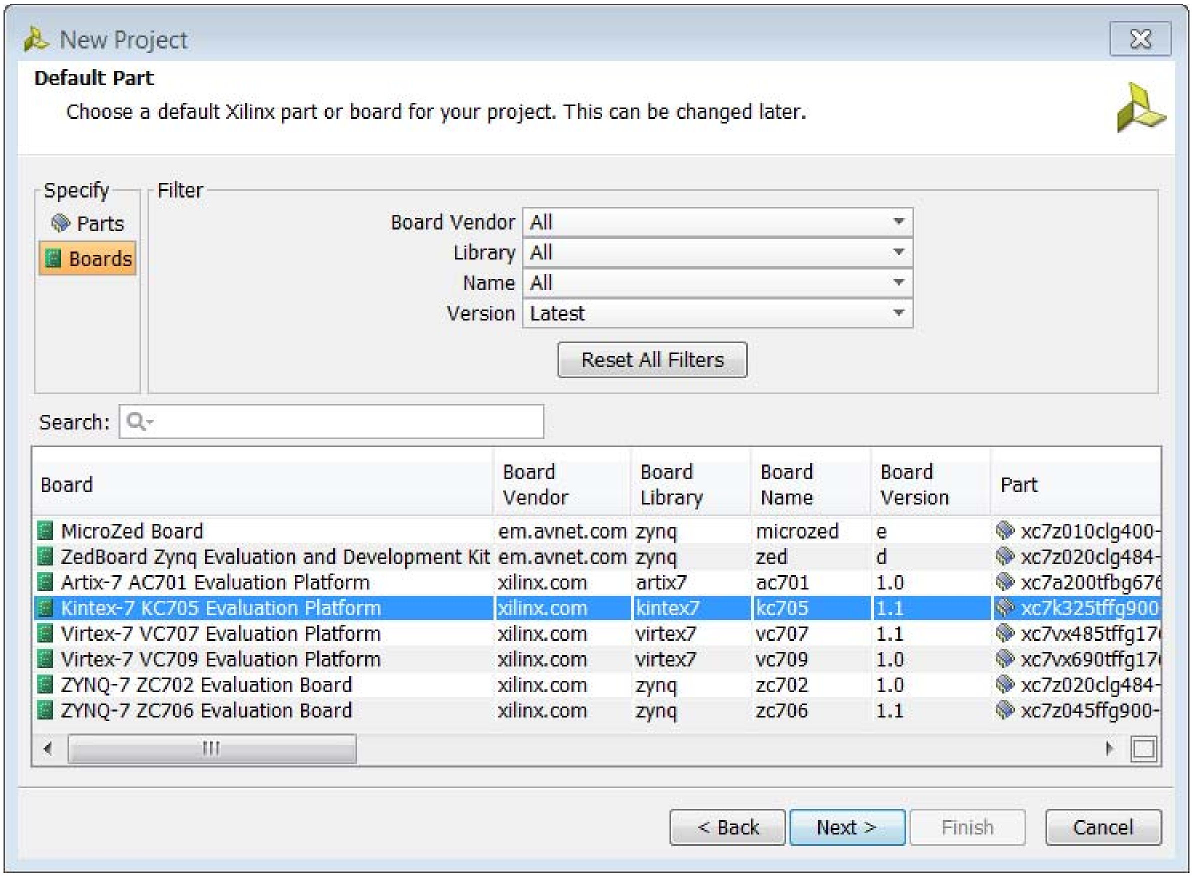
Further assistance adopting Vivado flows.

Release Notes Guide

Information about installation and new IDS features in this release.

Figure 4-1: **Create New Project Wizard Button**

Select the KC705 board as the target board.



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Board Vendor: All
Library: All
Name: All
Version: Latest

Search:

Board	Board Vendor	Board Library	Board Name	Board Version	Part
MicroZed Board	em.avnet.com	zynq	microzed	e	xc7z010clg400-
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	zynq	zed	d	xc7z020clg484-
Artix-7 AC701 Evaluation Platform	xilinx.com	artix7	ac701	1.0	xc7a200tfbg676
Kintex-7 KC705 Evaluation Platform	xilinx.com	kintex7	kc705	1.1	xc7k325tffg900
Virtex-7 VC707 Evaluation Platform	xilinx.com	virtex7	vc707	1.1	xc7vx485tffg17
Virtex-7 VC709 Evaluation Platform	xilinx.com	virtex7	vc709	1.0	xc7vx690tffg17
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	zynq	zc702	1.0	xc7z020clg484-
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	zynq	zc706	1.1	xc7z045ffg900-

Figure 4-2: **Select the Target Board**

Note: You can perform the same actions using the following Tcl commands:

```
create_project xx <your_directory>/xx -part xc7k325tffg900-2
set_property board xilinx.com:kintex7:kc705:1.0 [current_project]
set_property target_language VHDL [current_project]
```

In Tcl commands, angle brackets (<>) are used to enclose parameters specific to your design. Do not include the angle brackets in the command string.

Designing in IP Integrator

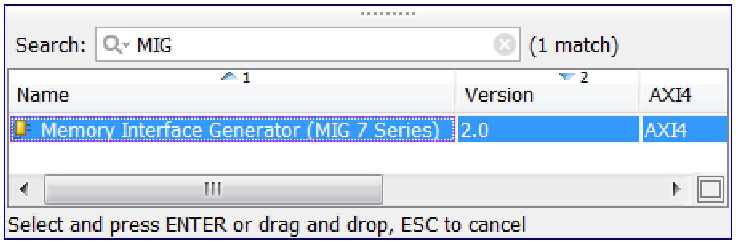
Create a new block design in the Flow Navigator by clicking **Create Block Design** under **IP Integrator**.

Tcl Command:

```
create_bd_design <your_design_name>
```

Adding the MIG IP

To add the memory interface generator IP, right-click in the diagram and select **Add IP**. A searchable IP Catalog opens. When you type the first few letters of the IP name (**`mig`**), only the IP matching the string is displayed.



Search: MIG (1 match)

Name	Version	AXI4
Memory Interface Generator (MIG 7 Series)	2.0	AXI4

Select and press ENTER or drag and drop, ESC to cancel

Figure 4-3: **Searching for MIG IP by Name**

Alternatively, you can click the **Add IP** button on the left side of the canvas .

This places the MIG IP core on the IP integrator block design.

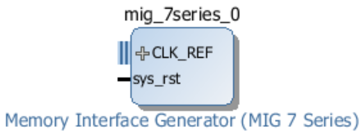


Figure 4-4: **MIG IP Core**

Use the same procedure to add a MicroBlaze™ or Zynq processor to the IP integrator block design.

Making Connections

Because the MIG core provides the clocking for the entire KC705 board, you run block automation for the MIG core before running it for a controller.

1. Click **Run Block Automation**, and then click **/mig_7series_0**.

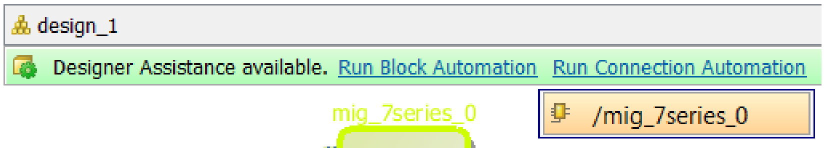

design_1
Designer Assistance available. Run Block Automation Run Connection Automation
mig_7series_0
/mig_7series_0

Figure 4-5: **Run Block Automation for the MIG Core**

This opens the Run Block Automation dialog box.

2. Click **OK**.

The following is generated in the block design. The MIG core is configured for 400 MHz operation with the correct pins selected for the KC705 board.

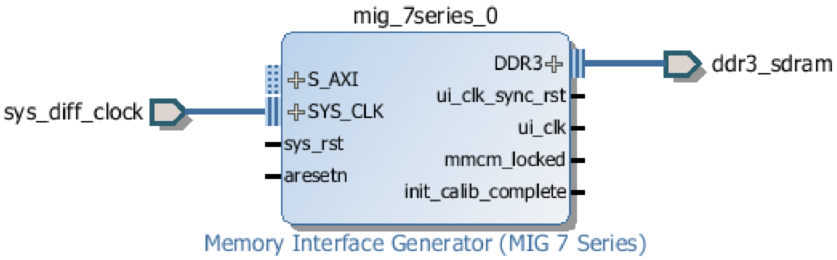


Figure 4-6: **MIG Block**

3. To make changes to the MIG configuration, right-click the block and then click **Customize Block**. Alternatively, you can double-click the MIG IP block.

More information on MIG configuration settings can be found in the *7 Series FPGAs Memory Interface Solutions User Guide* (UG586) [Ref 9].

Adding a Clocking Wizard with the MIG Core

If the design requires clocking in addition to that generated by the MIG core, you need to add a Clocking wizard IP in the block design.

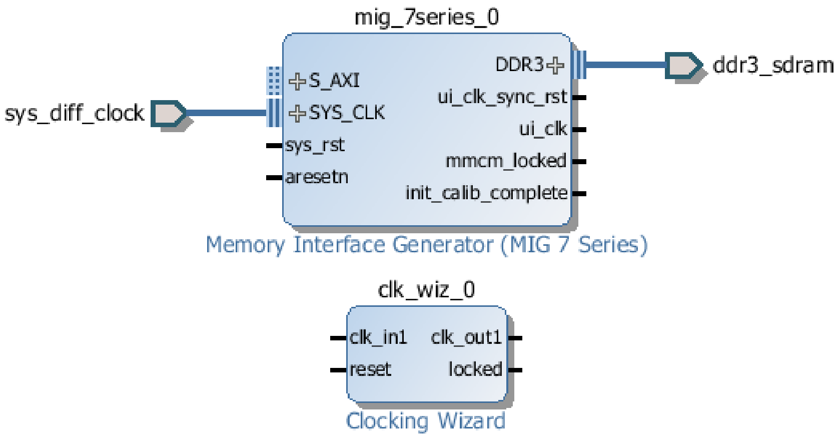


Figure 4-7: **Clocking Wizard**

Then, follow these steps to connect the Clocking Wizard to the MIG core.

1. Connect the `ui_clk` (or any of the additional clocks generated) output of the MIG core to the `clk_in1` input of the Clocking wizard.

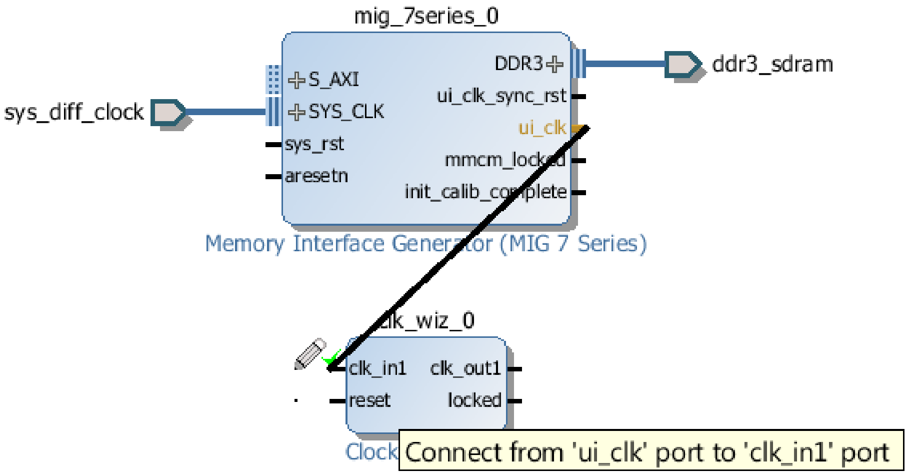


Figure 4-8: **Connect ui_clk to clk_in1**

2. Connect the `ui_clk_sync_rst` port of the MIG core to the reset port of the Clocking wizard.

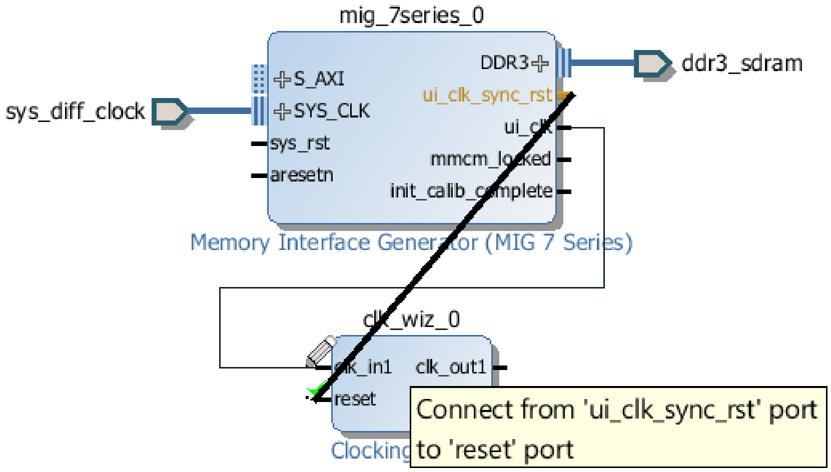


Figure 4-9: **Connect ui_clk_sync_rst to the reset Port**

3. Customize the Clocking wizard by double-clicking it to generate the required clocks for the design.

Instantiating an AXI Master in the IP Integrator Design

To complete the MIG design, a master such as a Zynq or MicroBlaze processor or an external processor is required.

The following procedure shows how to instantiate a MicroBlaze processor in the IP integrator design.

1. Add a MicroBlaze processor to the block design by right-clicking in the canvas and selecting **Add IP**.

   Figure 4-10 shows the processor instantiated on the block design.

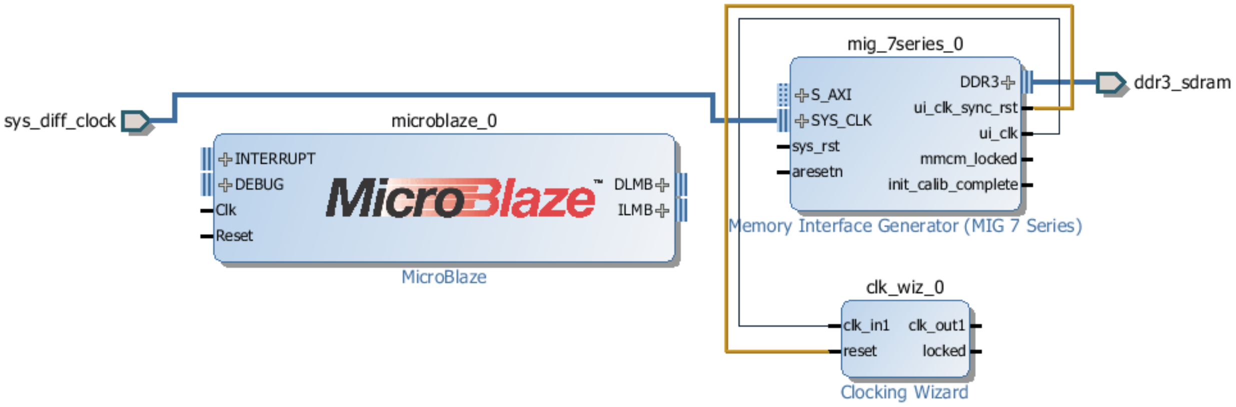
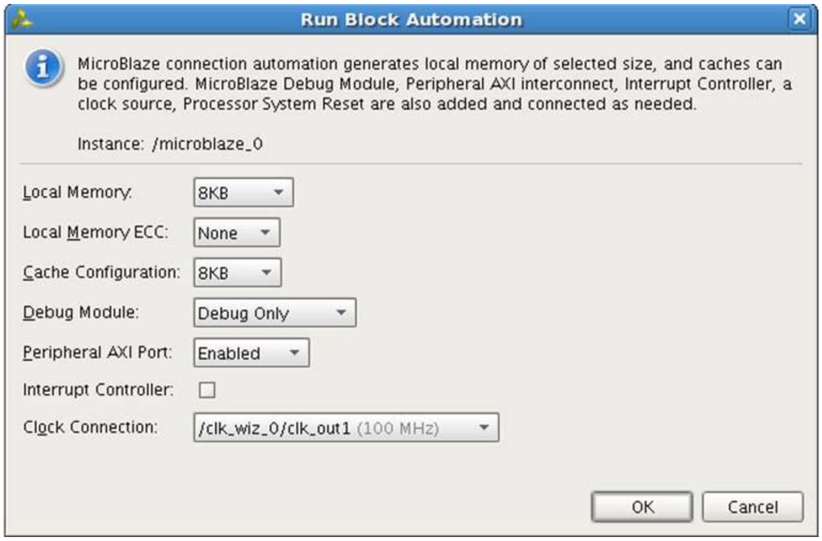


Figure 4-10: **Add a MicroBlaze Processor**

2. Click **Run Block Automation** to construct a basic MicroBlaze system and configure the settings:
   - Local Memory: Select the required amount of local memory from pull-down menu.
   - Local Memory ECC: Turn on ECC if desired.
   - Cache Configuration: Select the required amount of Cache memory.
   - Debug Module: Specify the type of debug module from the pull-down menu.
   - Peripheral AXI Interconnect: This option must be selected.
   - Interrupt Controller: Optional.
   - Clock Connection: Select the clock source from the pull-down menu.



Run Block Automation

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Local Memory: 8KB
Local Memory ECC: None
Cache Configuration: 8KB
Debug Module: Debug Only
Peripheral AXI Port: Enabled
Interrupt Controller: □
Clock Connection: /clk_wiz_0/clk_out1 (100 MHz)

Figure 4-11: **Run Block Automation Settings**

3. Click **OK**.

   The block design looks like Figure 4-12.

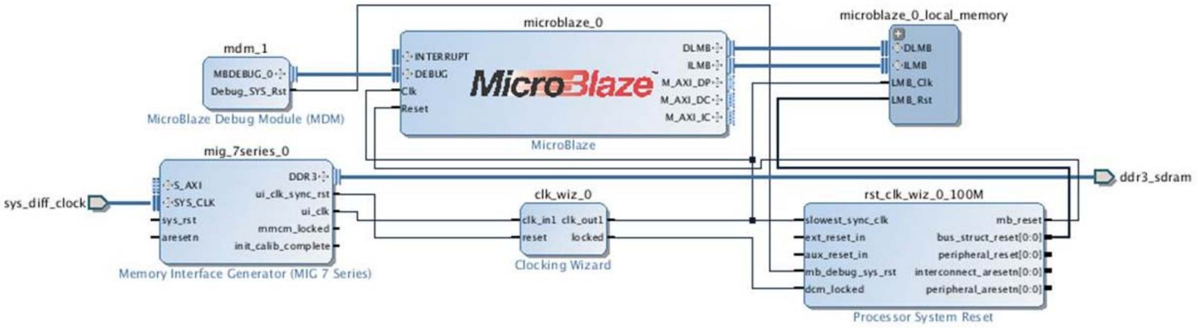
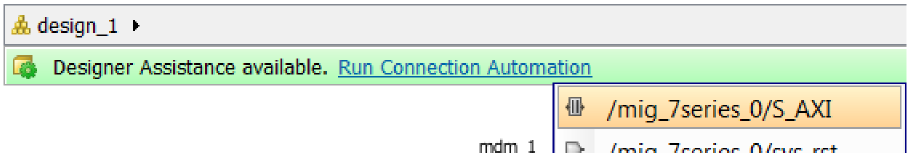


Figure 4-12: **Block Design with MicroBlaze Processor**

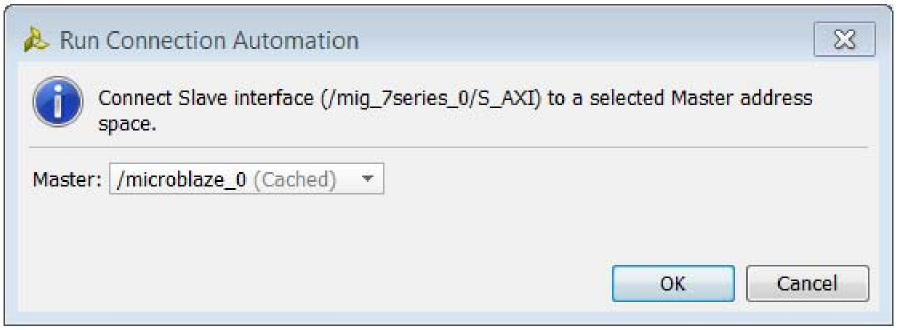
4. Click **Run Connection Automation** and connect the MIG core to the MicroBlaze processor.



Designer Assistance available. Run Connection Automation

Figure 4-13: **Run Connection Automation**

The Run Connection Automation dialog box opens.



Connect Slave interface (/mig_7series_0/S_AXI) to a selected Master address space.

Master: /microblaze_0 (Cached)

Figure 4-14: **Instantiate an AXI Interconnect**

5. Select the Cached option from the pull-down menu and click **OK**.

   This instantiates another AXI Interconnect and makes the required connection between the MIG core and the MicroBlaze processor.

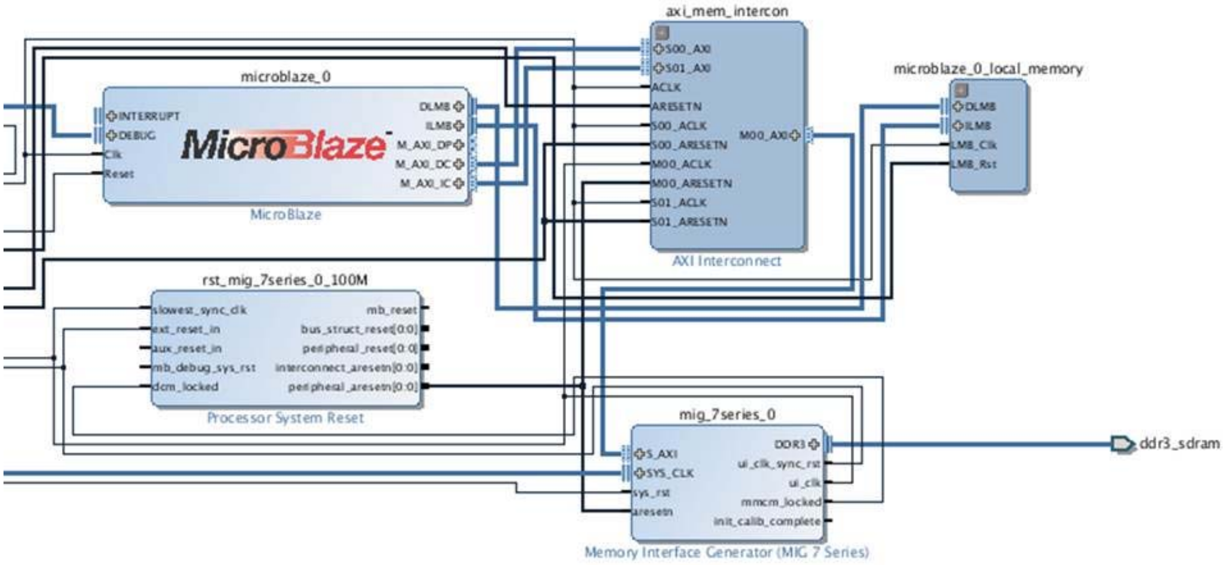


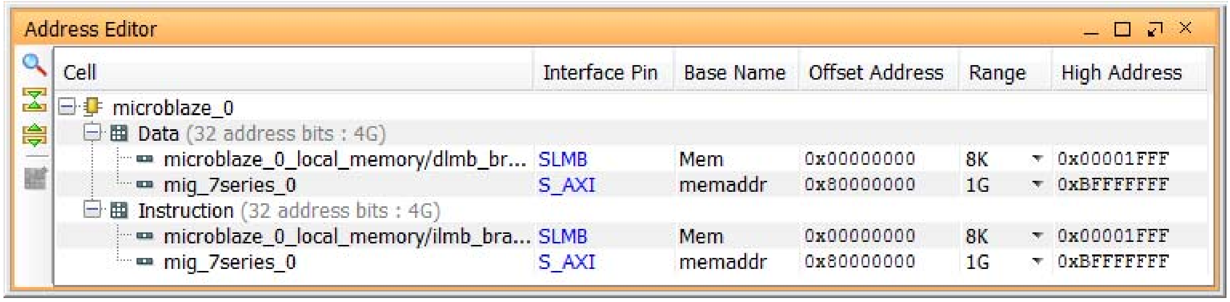
Figure 4-15: **MIG/MicroBlaze Connections**

6. Complete the remaining connections to the MIG core, such as to an external reset source.
7. Connect any interrupt sources through the Concat IP to the MicroBlaze processor.

Creating a Memory Map

To generate the address map for this design, click the **Address Editor** tab above the diagram. The memory map is automatically created as IP and added to the design. You can set the addresses manually by entering values in the **Offset Address** and **Range** columns.

Note: The Address Editor tab only appears if the diagram contains an IP block that functions as a bus master, such as the MicroBlaze processor in the following diagram.


Address Editor

Cell	Interface Pin	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_local_memory/dlmb_br...	SLMB	Mem	0x00000000	8K	0x00001FFF
mig_7series_0	S_AXI	memaddr	0x80000000	1G	0xBFFFFFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bra...	SLMB	Mem	0x00000000	8K	0x00001FFF
mig_7series_0	S_AXI	memaddr	0x80000000	1G	0xBFFFFFFF

Figure 4-16: **Address Editor**

Running Design Rule Checks

The Vivado IP integrator runs basic design rule checks in real time as you create the design. However, problems can occur during design creation. For example, the frequency on a clock pin might not be set correctly. To run a comprehensive design check, click the **Validate Design** button ☑.

If the design is free of warnings and errors, a successful validation dialog box displays.

Implementing the Design

Now you can implement the design, generate the bitstream, and create the software application in SDK.

Chapter 5

Reset and Clock Topologies in IP Integrator

Overview

This chapter provides information about clock and reset connectivity at the system level. In the Vivado® IP integrator, you can use the new Board Automation Flow (referred to hereafter as Board Flow), which enables you to configure IP at a board interface in an automated manner, or you can make all the connections manually.

To create designs with IP integrator that function correctly on the target hardware, you must understand reset and clocking considerations. The examples and overall flow in this chapter use the Board Flow, but the considerations apply equally for designs created without using the Board Flow.

In the IP integrator, the Memory Interface Generator (MIG) core is a clock source, and the primary clock from the board oscillator must be connected directly to the MIG core. The MIG core can generate up to five additional clocks, which you can use for resetting the design as needed. For designs that contain a MIG core, ensure that the primary onboard clock is connected to MIG, and then use the user clock (`ui_clock`) as additional clock sources for the rest of the design.

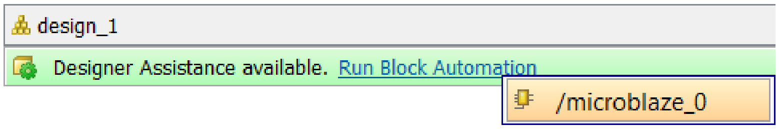
For IP integrator designs with Board Flow, specific IP (for example, MIG and Clocking wizard) support board-level clock configuration. For the rest of the system, clocking can be derived from the supported IP. Similarly, for driving reset signals, board-level reset configuration is supported by a specific reset IP (for example, proc_sys_reset). You can use other IP that also require external reset but are not currently supported by Board Flow.

The following sections illustrate the reset topologies for various types of designs.

MicroBlaze Design without a MIG Core

For any design that uses a MicroBlaze™ processor without a MIG core, you can instantiate a Clocking wizard to generate the clocks required. For Board Flow, you can configure the connection as follows:

1. After instantiating a MicroBlaze processor in the design, run Block Automation for MicroBlaze. This creates the MicroBlaze sub-system, as shown in Figure 5-1.



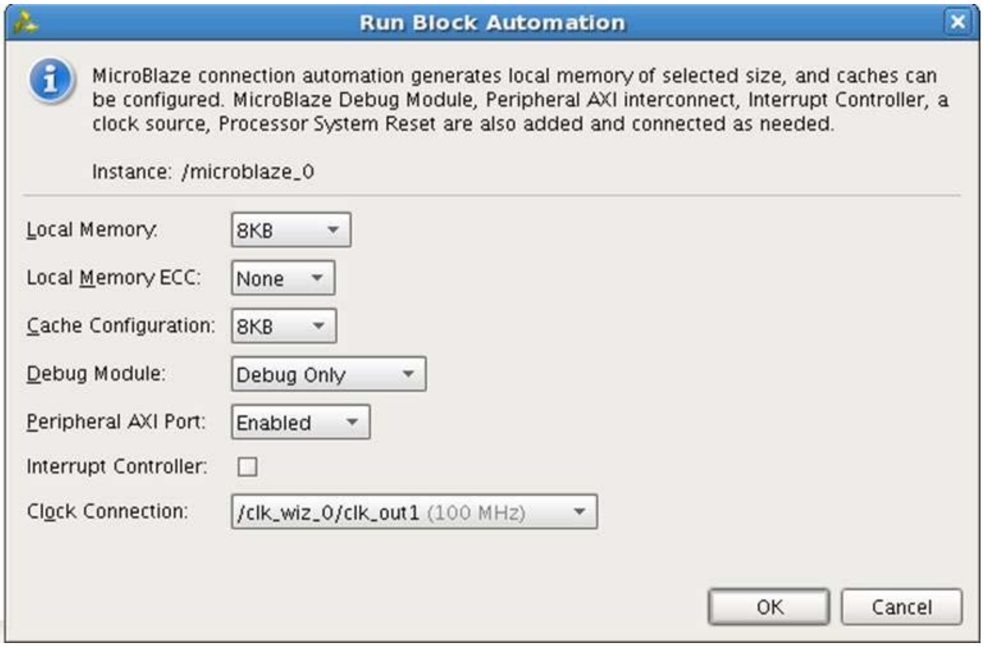
design_1

Designer Assistance available. Run Block Automation

/microblaze_0

Figure 5-1: **Run Block Automation on the MicroBlaze**

2. In the Run Block Automation dialog box, select the **New Clocking Wizard** option to instantiate the Clocking wizard IP, and click **OK**.



Run Block Automation

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Local Memory: 8KB

Local Memory ECC: None

Cache Configuration: 8KB

Debug Module: Debug Only

Peripheral AXI Port: Enabled

Interrupt Controller: □

Clock Connection: /clk_wiz_0/clk_out1 (100 MHz)

Figure 5-2: **Run Block Automation Dialog Box**

Running Block Automation also instantiates and connects the Proc Sys Reset IP to the various blocks in the design. The IP integrator canvas looks like Figure 5-3.

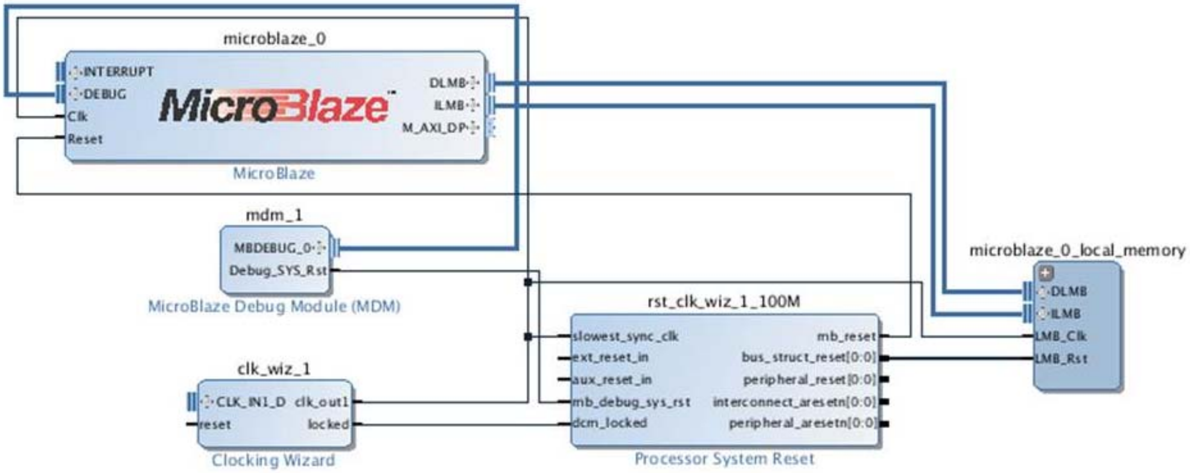
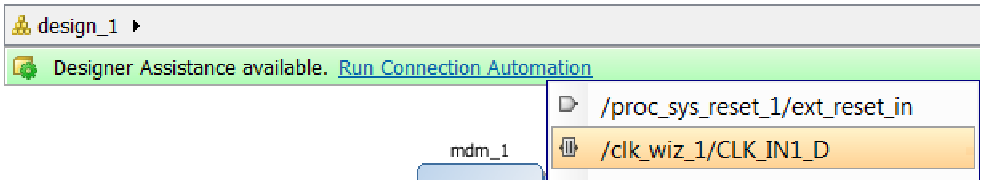


Figure 5-3: **Effect of Running Block Automation**

3. Click **Run Connection Automation** and select **/clk_wiz_1/CLK_IN1_D** to connect the on-board clock to the input of the Clocking Wizard IP, according to the board definition.

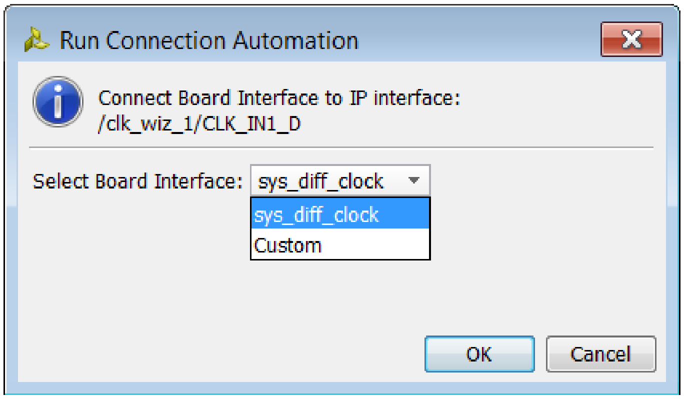
   Note: You can customize the Clocking Wizard to generate the various clocks required by the design.



Designer Assistance available. Run Connection Automation

Figure 5-4: **Running Connection Automation on the Clocking Wizard**

4. In the Run Connection Automation dialog box, select **sys_diff_clock** to select the board interface for the target board, or select **Custom** to tie a different input clock source to the Clocking Wizard IP, and then click **OK**.



Connect Board Interface to IP interface:
/clk_wiz_1/CLK_IN1_D

Select Board Interface: sys_diff_clock

Figure 5-5: **Connecting the On-board sys_diff_clock to the Clocking Wizard**

This creates a `sys_diff_clock` input port on the IP integrator canvas and then connects the port to the CLK_IN1_D input of the Clocking wizard.

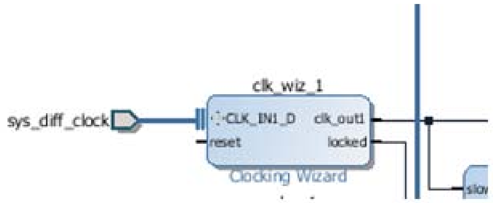
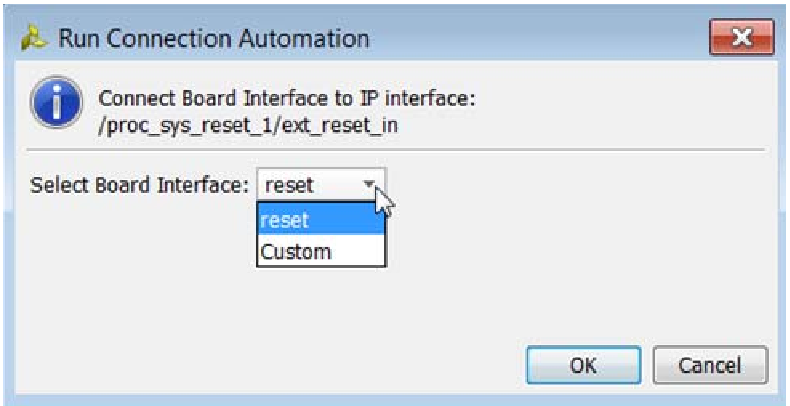


Figure 5-6: **Connecting the sys_diff_clock Input as the Source Clock to the Clocking Wizard**

5. Click **Run Connection Automation** and select **/proc_sys_reset_1/ext_reset_in** to connect the on-board reset to the input of `Proc_Sys_Reset`.
6. In the Run Connection Automation dialog box, select the dedicated reset interface on the target board or a Custom reset input source.


Run Connection Automation

Connect Board Interface to IP interface:
/proc_sys_reset_1/ext_reset_in

Select Board Interface: reset

Figure 5-7: **Connect the On-board Reset**

After you make your choice and click **OK**, the IP integrator canvas looks like Figure 5-8.

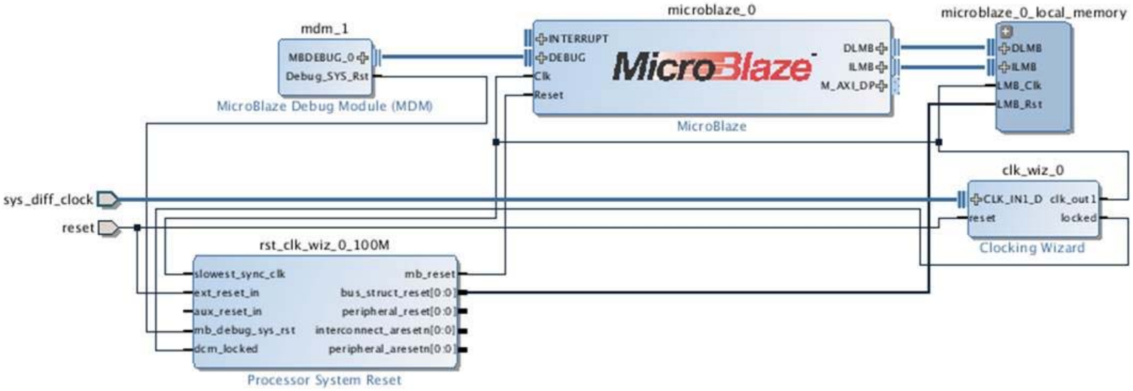


Figure 5-8: **On-Board Reset Connected to the Proc Sys Reset IP**

7. Click **Run Connection Automation** again and select **/clk_wiz_1/reset** to connect the reset input of the Clocking Wizard to the input reset source.

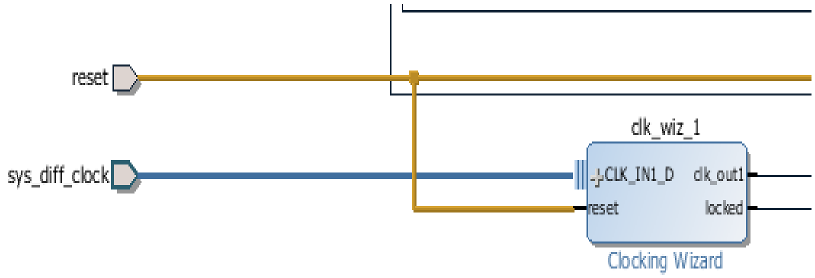


Figure 5-9: **Connecting the On-Board Reset to the Clocking Wizard**

CAUTION! *If Board Flow is not used, ensure that the "locked" output of the Clocking Wizard is connected to the "dcm_locked" input of Proc_Sys_Reset.*

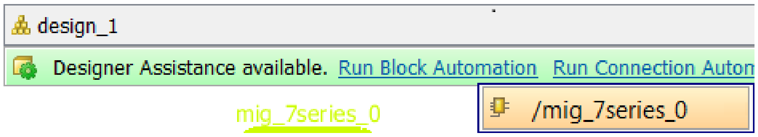
MicroBlaze Design with a MIG Core

RECOMMENDED: *As mentioned in the introduction, the MIG IP is a clock source, and Xilinx recommends that you connect the on-board clock directly to the MIG core.*

The MIG core provides a user clock (ui_clock) and up to five additional clocks that can be used in the rest of the design. You can configure the connection as follows:

1. When using Board Flow automation in a design that contains the MIG IP, Xilinx recommends that you add the MIG IP first and then run Block Automation. This connects the on-board clock to the MIG core.

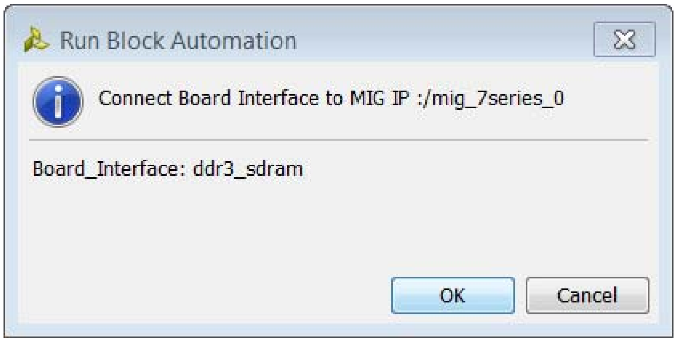
   You can then customize MIG to generate additional clocks, if required.


design_1

Designer Assistance available. Run Block Automation Run Connection Auton

Figure 5-10: **Running Block Automation on the MIG Core**

2. The Run Connection Automation dialog box states that the ddr3_sdram interface is available. Click **OK**.



Run Block Automation

Connect Board Interface to MIG IP :/mig_7series_0

Board_Interface: ddr3_sdram

Figure 5-11: **Running Block Automation on the MIG Core**

This connects the interface ports to the MIG as shown in Figure 5-12.

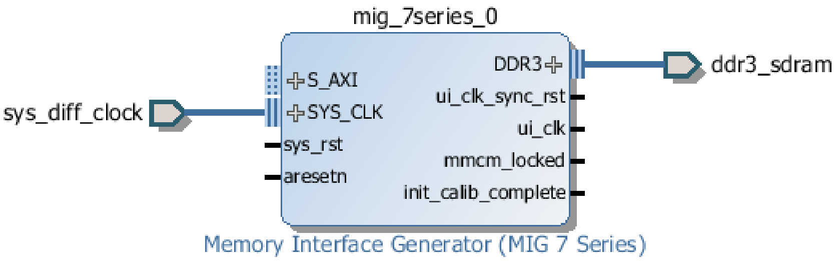
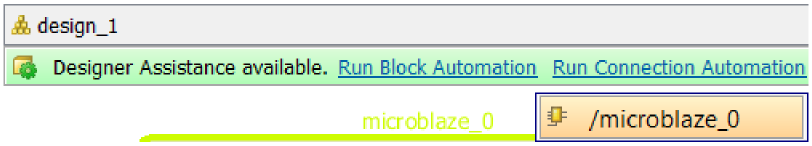


Figure 5-12: **Block Automation Creating the DDR3_SDRAM Interface**

3. Add the MicroBlaze processor to the design and run Block Automation.

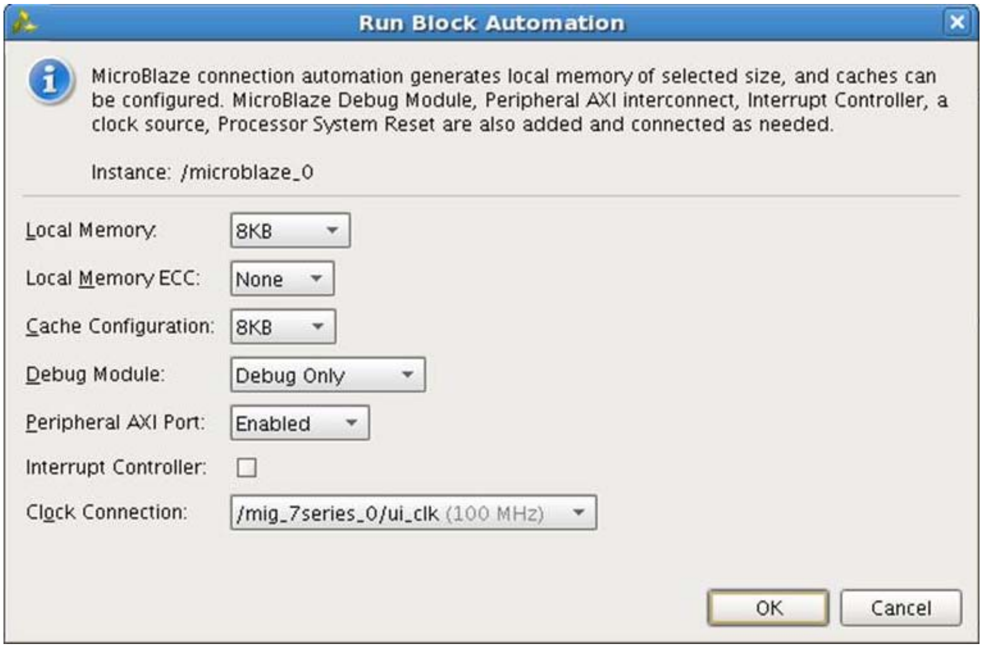


design_1

Designer Assistance available. Run Block Automation Run Connection Automation

Figure 5-13: **Instantiating and Running Block Automation on the MicroBlaze**

4. In the **Clock Connection** field of the Run Block Automation dialog box, select the MIG ui_clk (**/mig_7series_1/ui_clk**) as the clock source for the MicroBlaze processor, and then click **OK**.



Run Block Automation

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Local Memory: 8KB

Local Memory ECC: None

Cache Configuration: 8KB

Debug Module: Debug Only

Peripheral AXI Port: Enabled

Interrupt Controller: □

Clock Connection: /mig_7series_0/ui_clk (100 MHz)

Figure 5-14: **Run Block Automation Options for the MicroBlaze Processor**

This creates a MicroBlaze subsystem and connects the `ui_clk` as the input source clock to the subsystem, as shown by the highlighted net in Figure 5-15.

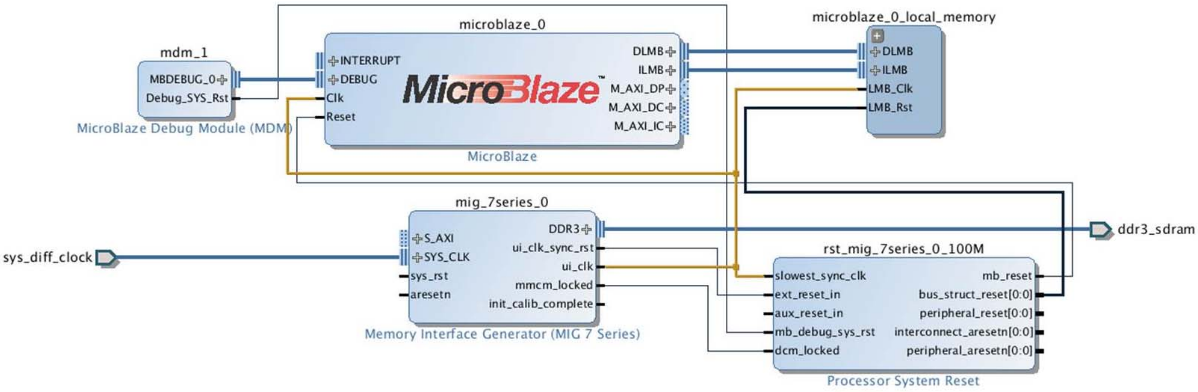
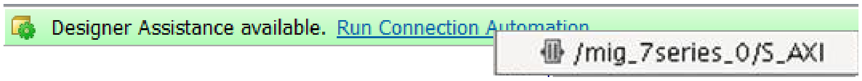


Figure 5-15: **Connect the Output Clock from the MIG Core to Clock the Design**

5. Make the following additional connections:

   a. Connect the on-board reset to the `sys_rst` input of the MIG IP.

   b. Click **Connection Automation** and select `/mig_7series/S_AXI` to connect the MIG to MicroBlaze.

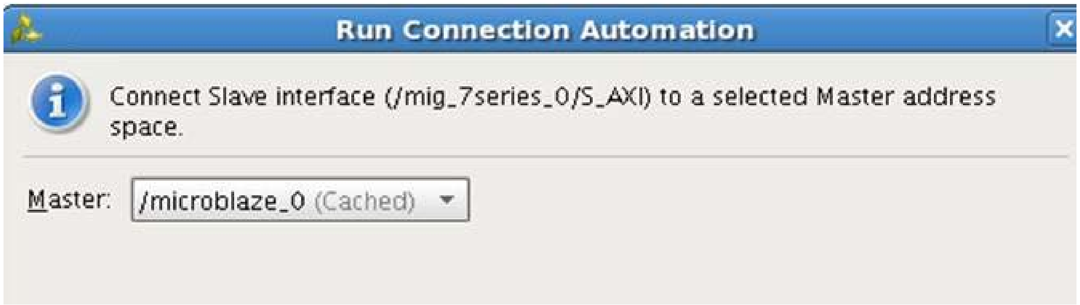


Designer Assistance available. Run Connection Automation

/mig_7series_0/S_AXI

Figure 5-16: **Running Connection Automation on mig_7series/S_AXI**

c. Select **/microblaze_0**.



Run Connection Automation

Connect Slave interface (/mig_7series_0/S_AXI) to a selected Master address space.

Master: /microblaze_0 (Cached)

Figure 5-17: **Run Connection Automation Dialog Box**

Figure 5-18 shows the completed connection for MB-MIG with Designer assistance.

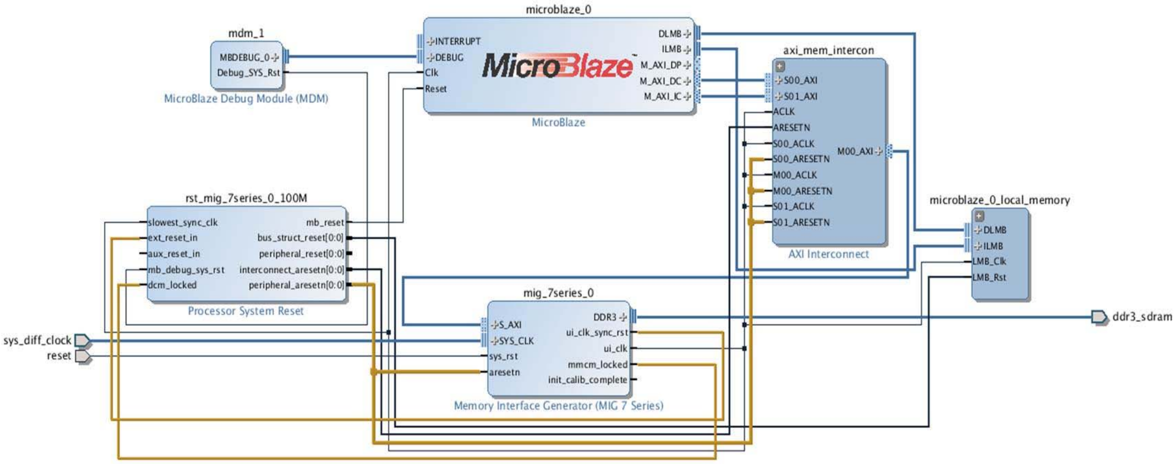


Figure 5-18: **Connect reset and mmcm_locked Pins**

Zynq Design without PL Logic

For Zynq designs without PL logic, all the clocks are contained in the ZYNQ7 Processing System IP. Use the following steps to add a Zynq design without PL.

1. After adding the ZYNQ7 Processing System IP, click **Run Block Automation** and select **/processing_system7_0**.

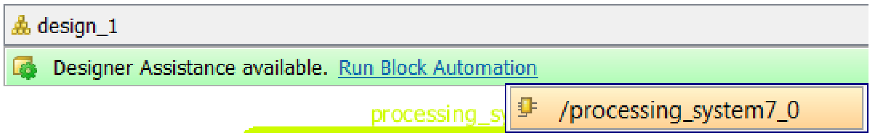
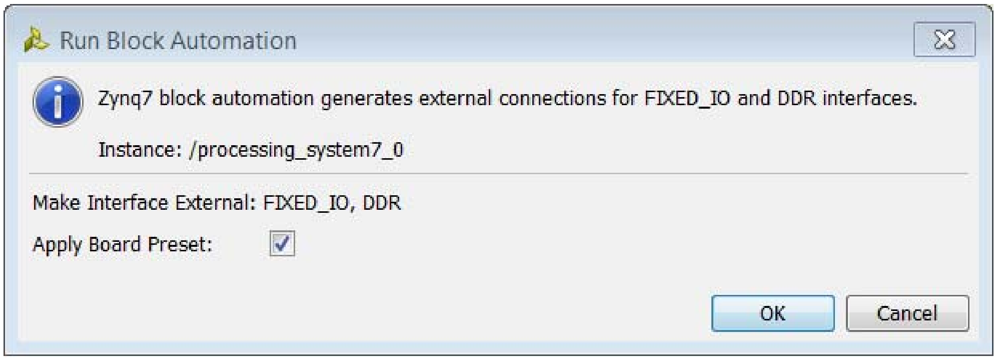


Figure 5-19: **Run Block Automation on Zynq**

2. The Run Block Automation states that the FIXED_IO and the DDR interfaces will be connected to external ports.
3. Click **OK**.



Zynq7 block automation generates external connections for FIXED_IO and DDR interfaces.

Instance: /processing_system7_0

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ☑

Figure 5-20: **Run Block Automation on the Zynq7 Processor**

4. Double-click the **ZYNQ7 Processing System** to re-customize the IP.

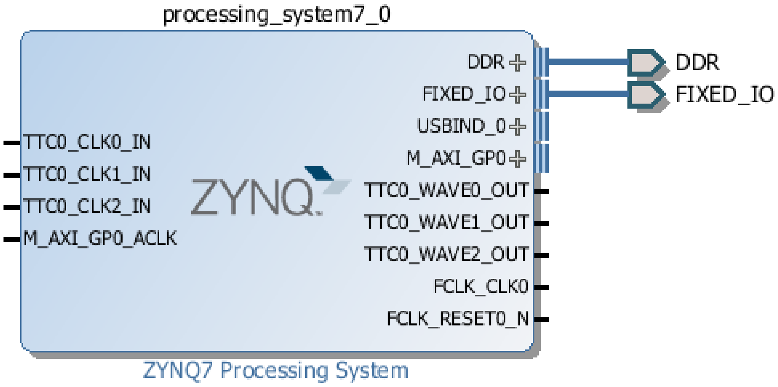
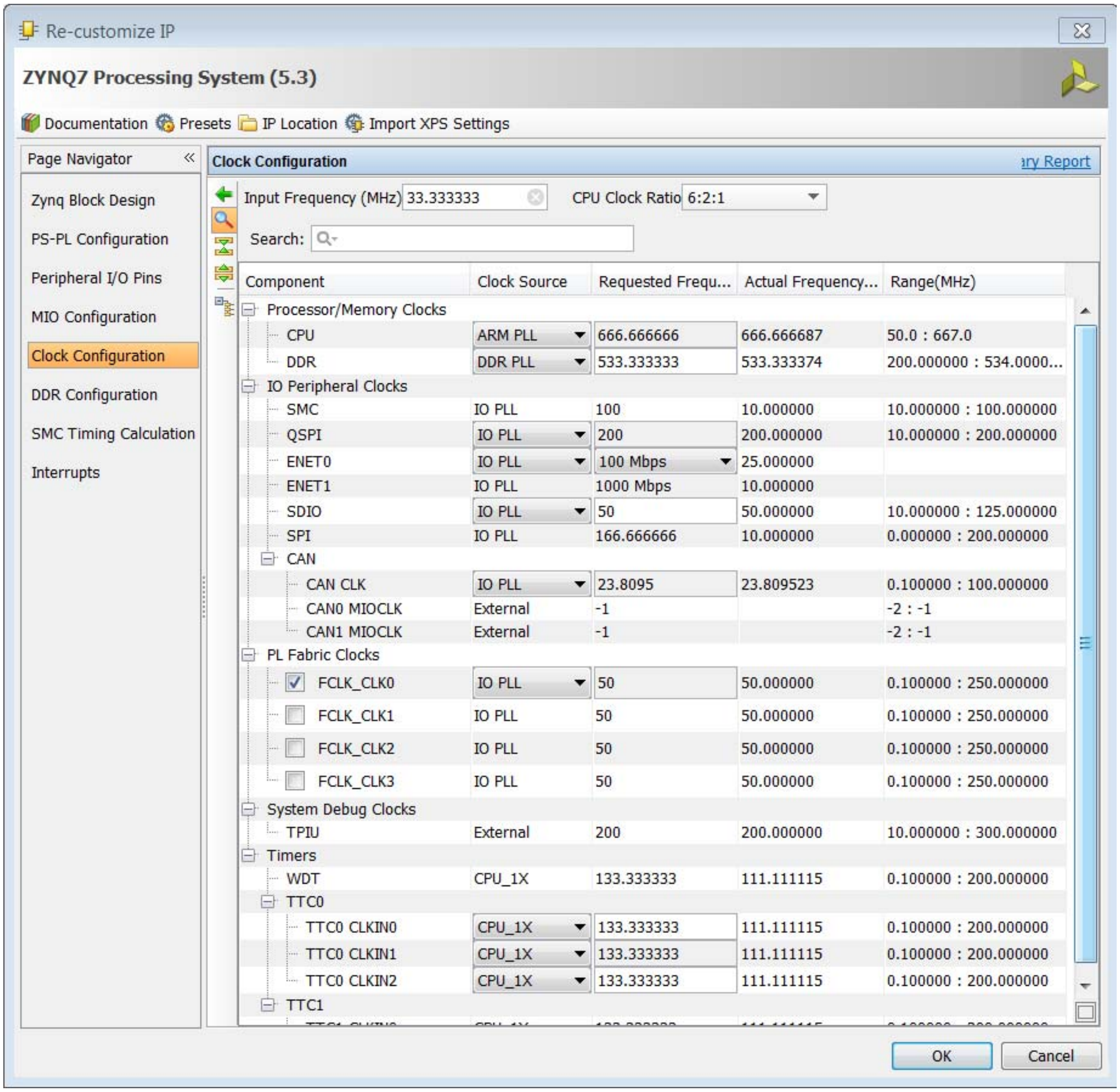


Figure 5-21: **Re-Customizing the Zynq IP**

5. Set the specific clocks in the Re-Customize_IP dialog box Clocking Configuration page.



Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequ...	Actual Frequency...	Range(MHz)
Processor/Memory Clocks				
CPU	ARM PLL	666.666666	666.666687	50.0 : 667.0
DDR	DDR PLL	533.333333	533.333374	200.000000 : 534.0000...
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	200.000000	10.000000 : 200.000000
ENET0	IO PLL	100 Mbps	25.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	50	50.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000
CAN				
CAN CLK	IO PLL	23.8095	23.809523	0.100000 : 100.000000
CAN0 MIOCLK	External	-1		-2 : -1
CAN1 MIOCLK	External	-1		-2 : -1
PL Fabric Clocks				
☑ FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK1	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 250.000000
System Debug Clocks				
TPIU	External	200	200.000000	10.000000 : 300.000000
Timers				
WDT	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0				
TTC0 CLKIN0	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN1	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC0 CLKIN2	CPU_1X	133.333333	111.111115	0.100000 : 200.000000
TTC1				

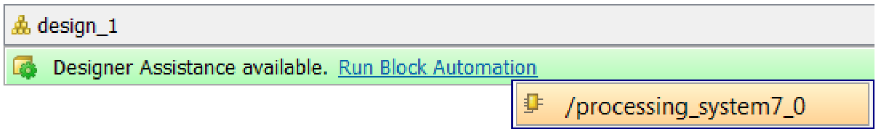
Figure 5-22: **Clock Configuration Options for the Zynq7 Processing System**

Zynq-7000 Design with PL Logic

RECOMMENDED: *For designs with a Zynq-7000 processor that contain custom logic in the PL fabric (but without MIG IP), it is recommended that the clocking and reset for the PL portion of the design be sourced from the PS. Any one of the PL Fabric Clocks-FCLK_CLK0, FCLK_CLK1, FCLK_CLK2 and FCLK_CLK3-can be used for the clock source. The associated resets for each of these clocks-FCLK_RESET0_N, FCLK_RESET1_N, FCLK_RESET2_N,and FCLK_RESET3_N-can be used for resetting the PL.*

Use the following steps to add a Zynq-7000 design with PL.

1. After adding the ZYNQ7 Processing System IP, click **Run Block Automation** and select **/processing_system7_0**.



design_1

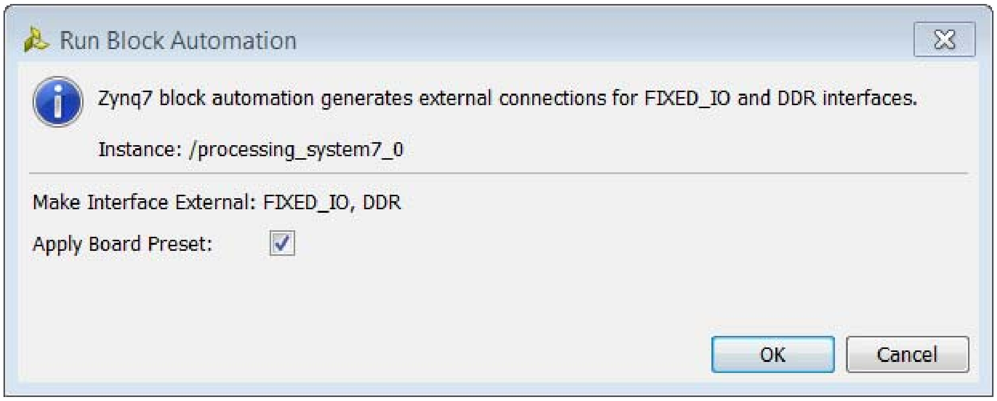
Designer Assistance available. Run Block Automation

/processing_system7_0

Figure 5-23: **Run Block Automation on the Zynq7 Processing System**

The Run Block Automation dialog box states that the FIXED_IO and the DDR interfaces will be connected to external ports.

2. Click **OK**.



Run Block Automation

Zynq7 block automation generates external connections for FIXED_IO and DDR interfaces.

Instance: /processing_system7_0

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ☑

Figure 5-24: **Run Block Automation Dialog Box for the Zynq7 Processing System**

3. Double-click the ZYNQ7 Processing System to re-customize the IP.

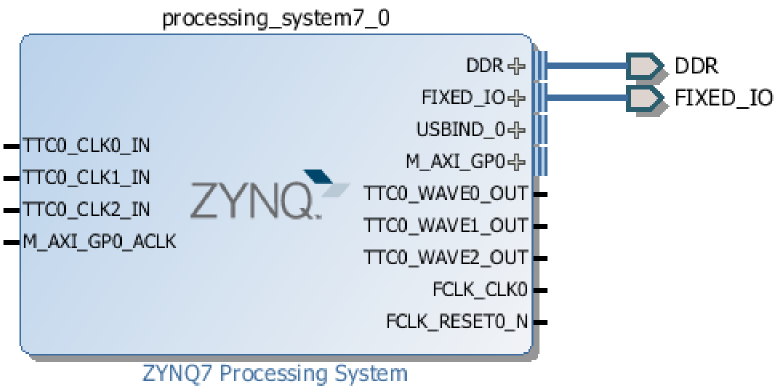
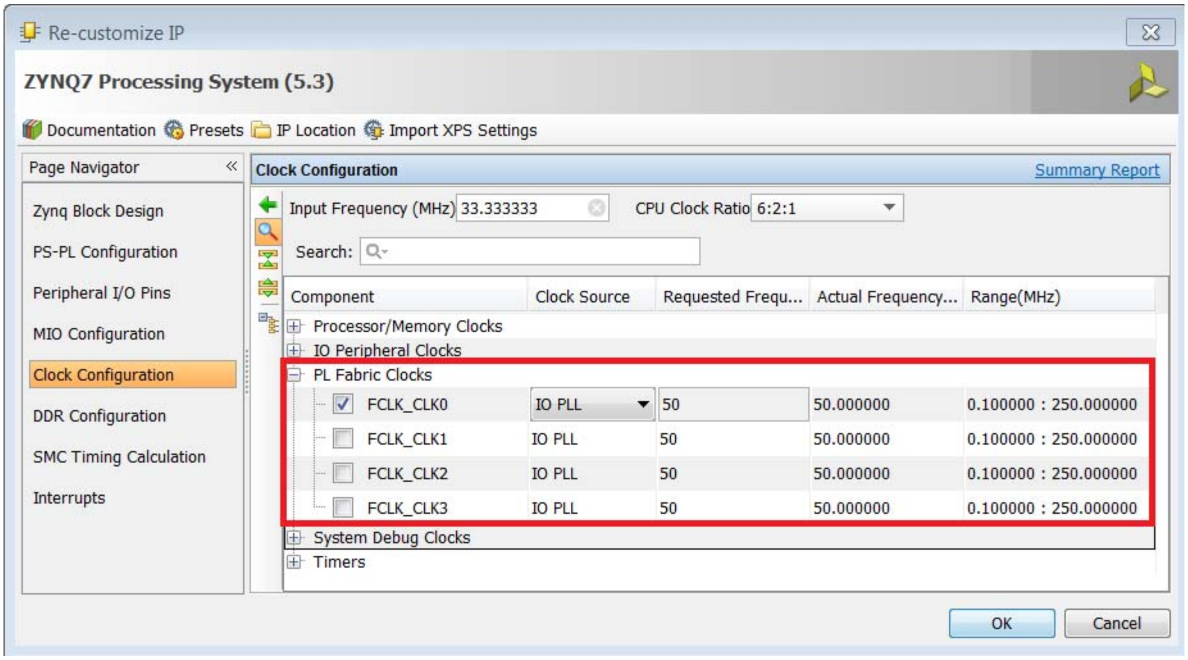


Figure 5-25: **Re-Customize the Zynq7 Processing System**

4. In the Re-customize IP dialog box, click **Clock Configuration** in the Page Navigator and then expand **PL Fabric Clocks**.



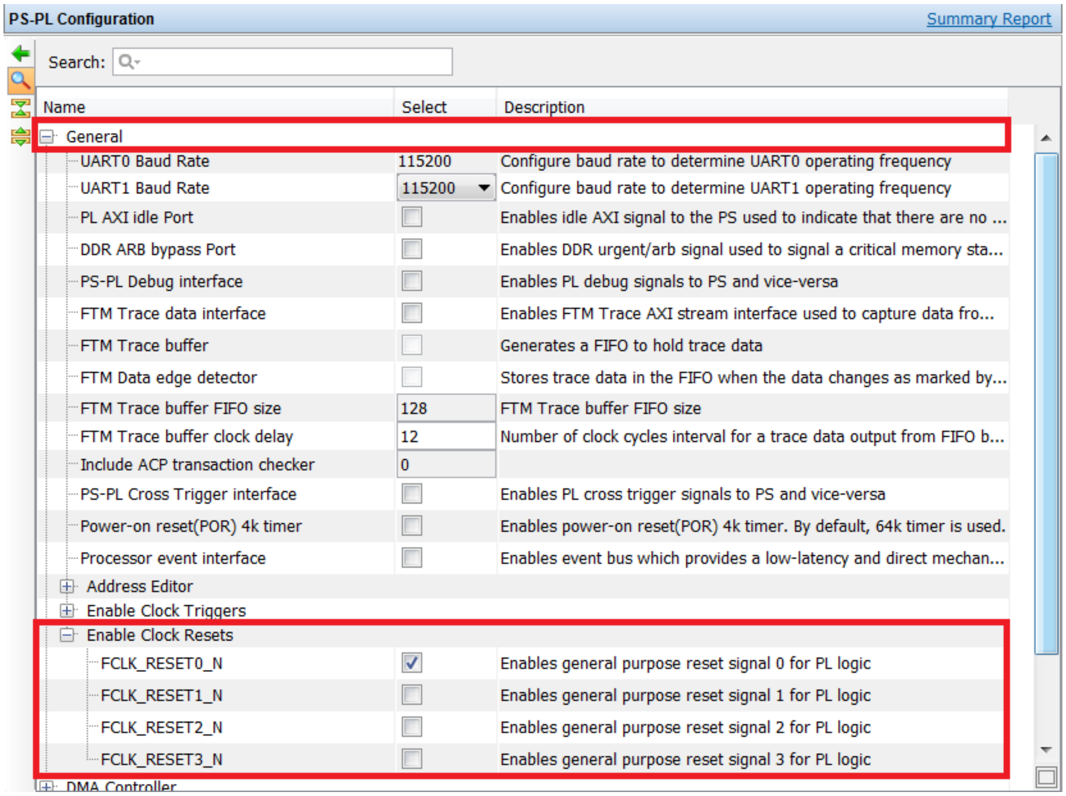
Input Frequency (MHz) 33.333333 CPU Clock Ratio 6:2:1

Component	Clock Source	Requested Frequ...	Actual Frequency...	Range(MHz)
⊞ Processor/Memory Clocks				
⊞ IO Peripheral Clocks				
⊟ PL Fabric Clocks				
☑ FCLK_CLK0	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK1	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 250.000000
□ FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 250.000000
⊞ System Debug Clocks				
⊞ Timers				

Figure 5-26: **Specify the Frequency of the Fabric Clock**

5. Click **PS-PL Configuration** in the Page Navigator and expand **General**.

6. Expand **Enable Clock Resets** and select the appropriate resets for the PL fabric.



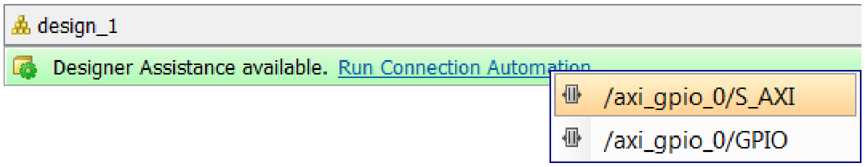
PS-PL Configuration	Summary Report

Search:

Name	Select	Description
General		
UART0 Baud Rate	115200	Configure baud rate to determine UART0 operating frequency
UART1 Baud Rate	115200	Configure baud rate to determine UART1 operating frequency
PL AXI idle Port	☐	Enables idle AXI signal to the PS used to indicate that there are no ...
DDR ARB bypass Port	☐	Enables DDR urgent/arb signal used to signal a critical memory sta...
PS-PL Debug interface	☐	Enables PL debug signals to PS and vice-versa
FTM Trace data interface	☐	Enables FTM Trace AXI stream interface used to capture data fro...
FTM Trace buffer	☐	Generates a FIFO to hold trace data
FTM Data edge detector	☐	Stores trace data in the FIFO when the data changes as marked by...
FTM Trace buffer FIFO size	128	FTM Trace buffer FIFO size
FTM Trace buffer clock delay	12	Number of clock cycles interval for a trace data output from FIFO b...
Include ACP transaction checker	0	
PS-PL Cross Trigger interface	☐	Enables PL cross trigger signals to PS and vice-versa
Power-on reset(POR) 4k timer	☐	Enables power-on reset(POR) 4k timer. By default, 64k timer is used.
Processor event interface	☐	Enables event bus which provides a low-latency and direct mechan...
Address Editor		
Enable Clock Triggers		
Enable Clock Resets		
FCLK_RESET0_N	☑	Enables general purpose reset signal 0 for PL logic
FCLK_RESET1_N	☐	Enables general purpose reset signal 1 for PL logic
FCLK_RESET2_N	☐	Enables general purpose reset signal 2 for PL logic
FCLK_RESET3_N	☐	Enables general purpose reset signal 3 for PL logic
DMA Controller		

Figure 5-27: **Specify the Output Clock to the PL Fabric**

7. Instantiate an IP such as AXI GPIO in the PL fabric. Then, click **Run Connection Automation** and select **/axi_gpio_1/s_axi**.



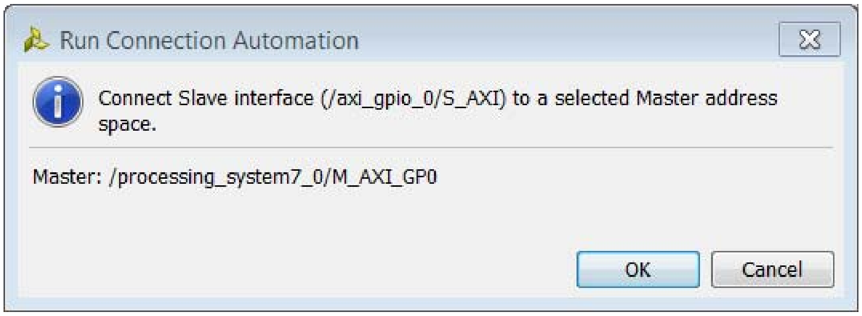
design_1

Designer Assistance available. Run Connection Automation

Figure 5-28: **Run Connection Automation on the GPIO Core**

The Run Connection Automation dialog box states that the s_axi port of the GPIO will be connected to the ZYNQ7 Processing System master.

8. Click **OK**.



Connect Slave interface (/axi_gpio_0/S_AXI) to a selected Master address space.

Master: /processing_system7_0/M_AXI_GP0

Figure 5-29: **Run Connection Automation Dialog Box to Connect GPIO**

The clock and resets in the IP integrator design should look as shown in the highlighted nets in Figure 5-30.

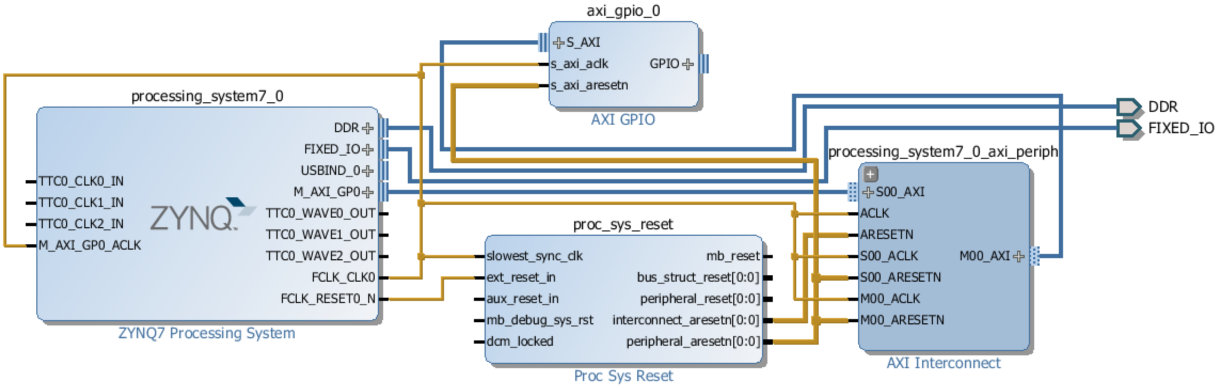
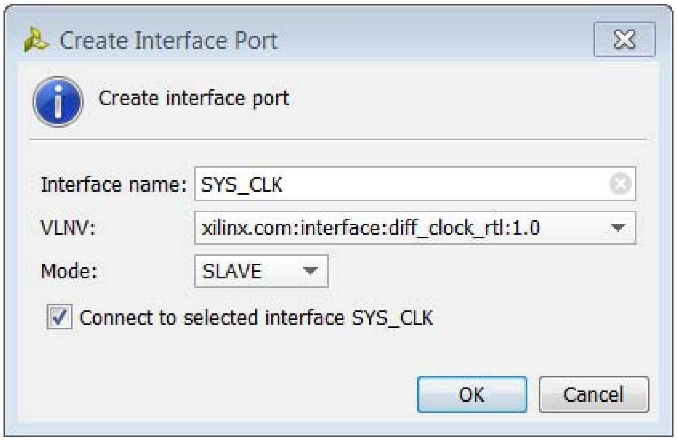


Figure 5-30: **Using the Output Clock from the Zynq PS7 IP to Clock the Design**

Zynq Design with a MIG core in the PL

RECOMMENDED: *For Zynq designs that include a MIG core in the PL, it is recommended that the input clock to the MIG core use an external clock source instead of the PS Fabric clock. The external clock from an on-board oscillator would be cleaner in terms of jitter when compared to clocks from the PS. You can use PS Fabric clocks for other portions of the PL design if required.*

1. Add the MIG IP and configure according to design requirements.
2. Then, connect the input clock source to the `SYS_CLK` input of the MIG core by right-clicking `SYS_CLK` in the block design and selecting **Create Interface Port**.
3. In the Create Interface Port dialog box, specify the options as shown in Figure 5-31.
4. Click **OK**.



Interface name: SYS_CLK

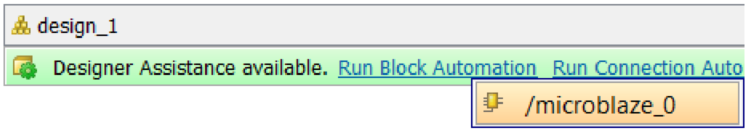
VLNV: xilinx.com:interface:diff_clock_rtl:1.0

Mode: SLAVE

☑ Connect to selected interface SYS_CLK

Figure 5-31: **Connecting the On-Board Clock Source to the MIG Core**

5. If the design uses a MicroBlaze processor, add it to the design and run MicroBlaze Block Automation.



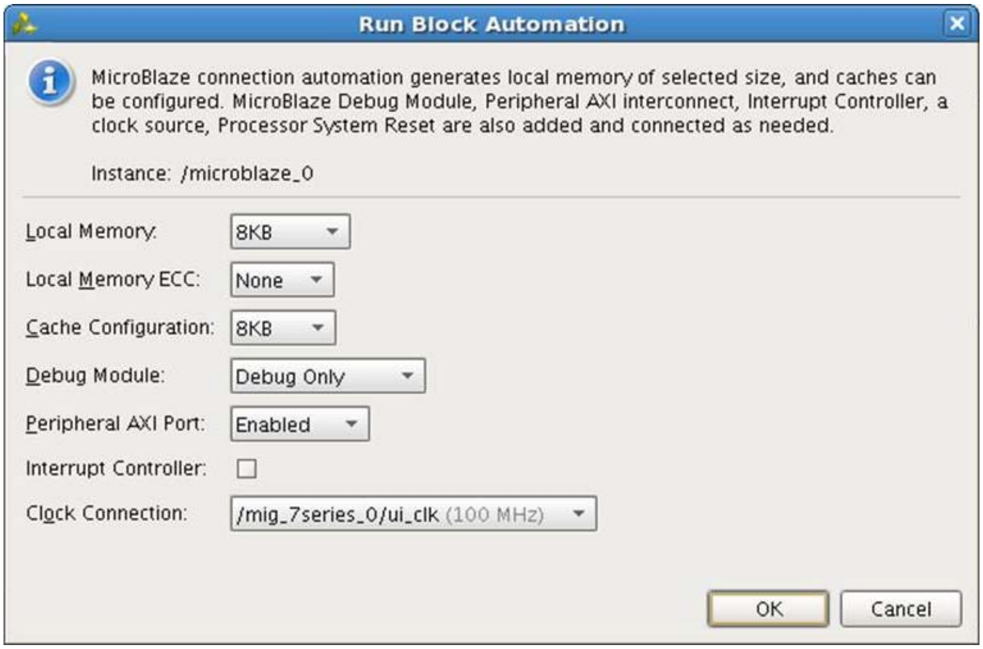
Designer Assistance available. Run Block Automation Run Connection Auto

Figure 5-32: **Running Block Automation on the MicroBlaze Processor**

In this case, you select the MIG `ui_clk` as the clock connection.

The Run Block Automation dialog box opens.

6. Specify **/mig_7series_1/ui_clk** as the input clock.



Run Block Automation

MicroBlaze connection automation generates local memory of selected size, and caches can be configured. MicroBlaze Debug Module, Peripheral AXI interconnect, Interrupt Controller, a clock source, Processor System Reset are also added and connected as needed.

Instance: /microblaze_0

Local Memory:	8KB
Local Memory ECC:	None
Cache Configuration:	8KB
Debug Module:	Debug Only
Peripheral AXI Port:	Enabled
Interrupt Controller:	□
Clock Connection:	/mig_7series_0/ui_clk (100 MHz)

Figure 5-33: **Specifying MicroBlaze Options**

7. Click **OK**.

The block design looks like Figure 5-34.

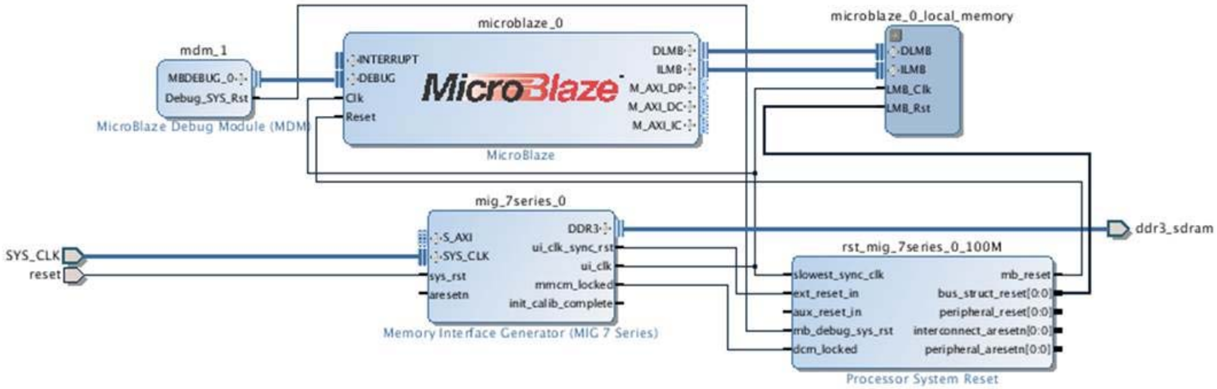


Figure 5-34: **Block Design after Running Block Automation**

The resulting changes look like Figure 5-35.

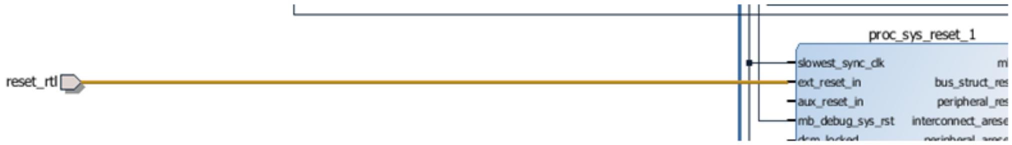


Figure 5-35: **Connect a Custom External Reset Source to the Design**

8. Connect the on-board reset to the `sys_rst` input of the MIG IP.

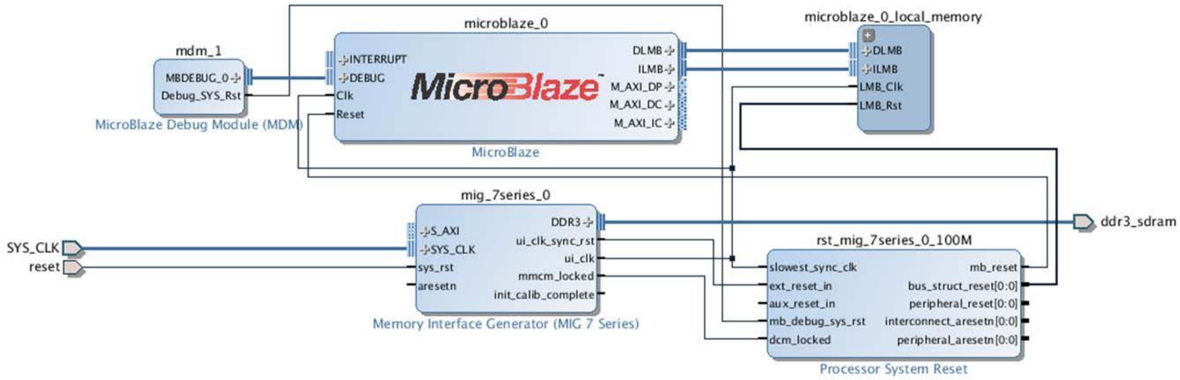


Figure 5-36: **Complete the Block Design**

Designs with MIG and the Clocking Wizard

For designs that require specific clock frequencies not generated by the MIG core, you can instantiate a Clocking Wizard IP and use the `ui_clock` output of the MIG IP as the clock input for the IP Clocking wizard.

You also need to make the following additional connections:

1. Connect the onboard reset to the Clocking Wizard reset input in addition to the MIG IP.
2. Connect the `mmcm_locked` pin of the MIG and locked pin of Clocking wizard to the `Util_Vector_Logic` IP configured to the AND operation. Then, connect the output of the `Util_Vector_Logic` to the `dcm_locked` input of `Proc_Sys_Reset`.

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see www.xilinx.com/company/terms.htm.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this guide:

1. *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821)
2. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
3. *Zynq-7000 AP SoC Technical Reference Manual* (UG585)
4. *Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator* (UG995)
5. *Zynq-7000 All Programmable SoC PCB Design and Pin Planning Guide* (UG933)
6. *MicroBlaze Processor Reference Guide* (UG081)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Using Constraints* (UG903)
9. *7 Series FPGAs Memory Interface Solutions User Guide* (UG586)